

# Towards the artificial evolution of target features in complex chemical systems

Peter Siepmann BSc (Hons), ARCO, LRSM

Thesis submitted to The University of Nottingham  
for the degree of Doctor of Philosophy, July 2010

# Contents

<b>List of Figures</b>	<b>iv</b>
<b>List of Tables</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and aim	4
1.2 Methodology	6
1.3 Contributions	7
1.4 Overview and structure	10
<b>2 Context and background - a literature review</b>	<b>12</b>
2.1 Self-organisation	13
2.2 Complex system design	15
2.3 Evolutionary Computation	16
2.4 Frameworks	24
2.5 Applications	25
2.6 Conclusions	26
<b>3 The Evolutionary Engine</b>	<b>27</b>
3.1 System architecture	28
3.2 Parellisation	35
3.3 Remote laboratory connection	35
3.4 Conclusions	41
<b>4 A complex search space analysis protocol</b>	<b>44</b>
4.1 Cellular automata - a description	45
4.2 Fitness	49
4.3 Robustness verification	51
4.4 Conclusions	58
<b>5 Evolving cellular automaton systems</b>	<b>60</b>
5.1 Experimental setup	60

5.2	Results	61
5.3	Local search	67
5.4	A further assessment of the USM	81
5.5	Conclusions	86
<b>6</b>	<b>Evolutionary design of nanostructures</b>	<b>88</b>
6.1	Nanostructures	89
6.2	Fitness	94
6.3	Robustness analysis	99
6.4	Results	103
6.5	Identifying limitations in Minkowski image analysis: Multi-phased nanostructures	111
6.6	Conclusions	115
<b>7</b>	<b>Accelerated evolution through fitness approximation</b>	<b>117</b>
7.1	Artificial neural networks	118
7.2	Artificial neural networks as fitness approximators	123
7.3	Results	127
7.4	Conclusions	128
<b>8</b>	<b>Conclusions and outlook</b>	<b>134</b>
	<b>References</b>	<b>139</b>
<b>A</b>	<b>Datasets</b>	<b>152</b>
A.1	Turbulence	152
A.2	Nanostructures	160

## List of Figures

2.1	Graphical representation of the uniform crossover method of parent selection in a genetic algorithm . . . . .	19
2.2	Graphical representation of the n-point crossover method of parent selection in a genetic algorithm . . . . .	19
2.3	Graphical representation of the uniform arithmetic crossover method of parent selection in a genetic algorithm . . . . .	20
2.4	Graphical representation of a Gaussian mutation operator . . . . .	21
2.5	Graphical representation of replacement strategies available for use within a genetic algorithm . . . . .	22
2.6	A graphical explanation of Pareto optimality . . . . .	23
3.1	An example XML configuration script used within the Evolutionary Engine . . . . .	29
3.2	Code outline of the Evolutionary Engine's problem specification class.	30
3.3	Diagrammatic representation of the logical structure and operation of the Evolutionary Engine . . . . .	31
3.4	Screenshot of the web-based configuration module within the Evolutionary Engine . . . . .	32
3.5	Screenshot of the web-based execution module within in the Evolutionary Engine . . . . .	33
3.6	Diagrammatic representation of the Evolutionary Engine's class structure	34
3.7	Graphical representation of the parellisation capabilities of the Evolutionary Engine. . . . .	37
3.8	Diagram and photograph of the chemical reactor array to which the Evolutionary Engine is designed to connect . . . . .	39
3.9	A schematic showing the communication protocol between the Evolutionary Engine and the remote chemical reactor array . . . . .	40
3.10	Evolution graph of the simple 'find a colour' experiment using the remote laboratory reactor array in conjunction with the Evolutionary Engine . . . . .	42

3.11	Absorbance spectra for generations 0-3 in the simple 'find a colour' experiment using the remote laboratory reactor array in conjunction with the Evolutionary Engine. . . . .	43
4.1	The elementary cellular automaton system produced using Wolfram's "Rule 30" . . . . .	47
4.2	A example pattern from the <i>Turbulence</i> cellular automaton model . .	49
4.3	Mappings and analysis methods . . . . .	52
4.4	FDC scatter plot for the <i>Turbulence</i> system. $r = 0.165$ . . . . .	55
4.5	FDC scatter plot for the <i>Turbulence</i> system ( <i>initial-turbulence</i> only). $r = 0.143$ . . . . .	56
4.6	FDC scatter plot for the <i>Turbulence</i> system ( <i>coupling-strength</i> only). $r = -0.375$ . . . . .	56
4.7	FDC scatter plot for the <i>Turbulence</i> system ( <i>roughness</i> only). $r = -0.151$	57
4.8	Logarithmic clustering tree showing the <i>Turbulence</i> dataset grouped according to phenotypic similarity (as calculated by the Universal Similarity Metric) . . . . .	59
5.1	Graph of fitness against time, plus target and evolved behaviour for a GA functioning on problem Turb-20-0.8-0.005 . . . . .	63
5.2	Graph of fitness against time, plus target and evolved behaviour for a GA functioning on Turb-20-0-0.005 . . . . .	64
5.3	Graph of fitness against time, plus target and evolved behaviour for a GA functioning on problem Turb-40-0.2-0.005 . . . . .	65
5.4	Graph of fitness against time, plus target and evolved behaviour for a GA functioning on problem Turb-60-0.4-0 . . . . .	66
5.5	Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-20-0.8-0.005 . . . . .	70
5.6	Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-20-0-0.005 . . . . .	71
5.7	Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-40-0.2-0.005 . . . . .	72
5.8	Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-60-0.4-0 . . . . .	73
5.9	Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-20-0.8-0.005 . . . . .	76
5.10	Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-20-0-0.005 . . . . .	77
5.11	Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-40-0.2-0.005 . . . . .	78
5.12	Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-60-0.4-0 . . . . .	79

5.13	Target patterns for meta-automaton group A (one rule per model) . .	82
5.14	Target patterns for meta-automaton group B (two rules per model) .	82
5.15	Target patterns for meta-automaton group C (four rules per model) .	83
5.16	Examples of mirror images evolved for meta-automaton targets (target, left; evolved, right): (a) A2, (b) A4, (c) B1, (d) B2, (e) B7 . . . . .	85
5.17	Examples of other similarities captured in evolved meta-automaton targets (target, left; evolved, right): (a) A9, (b) B8, (c) B9 . . . . .	85
5.18	Target and evolved patterns for meta-automaton group C (four rules) (target, left; evolved, right): (a) C1, (b) C2, (c) C3 . . . . .	85
6.1	3D atomic force microscope images demonstrating four types of commonly observed nanostructured patterns. Imaging by Christopher Martin/Philip Moriarty (University of Nottingham) . . . . .	89
6.2	Diagrammatic representation of the construction of a Thiol-passivated Gold nanoparticle . . . . .	90
6.3	Example morphologies obtained through the spin casting of Thiol-passivated Gold nanoparticles onto a silicon substrate . . . . .	91
6.4	Hierarchical clustering tree showing USM classification of the nano system dataset . . . . .	96
6.5	FDC ‘target’ individual (Parameters: $< 35, 0.21, 1, 3 >$ , Simulation time: 976s . . . . .	101
6.6	FDC scatter plot for the nano system (area only). $r = -0.137$ . . . . .	101
6.7	FDC scatter plot for the nano system (perimeter only). $r = 0.548$ . .	102
6.8	FDC scatter plot for the nano system (Euler only). $r = 0.556$ . . . .	102
6.9	Logarithmic clustering tree showing the classification of the nano system dataset by our Minkowski-based similarity metric . . . . .	104
6.10	Nanostructured target patterns, both ‘real’ atomic force microscope images, and thresholded, despeckled, binary version for use within the evolutionary algorithm . . . . .	105
6.11	Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “cell” target . . . . .	107
6.12	Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “worm” target . . . . .	108
6.13	Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “island” target . . . . .	109
6.14	Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “labyrinth” target . . . . .	110
6.15	Extended nanostructure evolution: Phase separation . . . . .	114
6.16	Evolved two-phased nanostructure . . . . .	115
7.1	Schematic of a biological neuron . . . . .	119
7.2	Neural network implementations of logic gates . . . . .	120

7.3	Graph of fitness against time for the evolution of the neural network approximating the Minkowski area of a nano simulator pattern . . . .	126
7.4	Graph of fitness against time for the evolution of the neural network approximating the Minkowski perimeter of a nano simulator pattern .	126
7.5	Graph of fitness against time for the evolution of the neural network approximating the Minkowski Euler characteristic of a nano simulator pattern . . . . .	127
7.6	Evolution of nanostructures with surrogate fitness - target “cell” . . .	129
7.7	Evolution of nanostructures with surrogate fitness - target “worm” . .	129
7.8	Evolution of nanostructures with surrogate fitness - target “island” .	130
7.9	Evolution of nanostructures with surrogate fitness - target “labyrinth”	130
7.10	Comparision of evolved patterns using a) the nano system simulator with Minkowski analysis and b) the neural network fitness approximation, along with total run-time statistics (hr:min:sec) . . . . .	132

## List of Tables

3.1	Customisable options and user-specified operators for the Evolutionary Engine . . . . .	36
5.1	Statistical analysis of Turbulence system evolutions, using GA/USM.	62
5.2	Statistical analysis of Turbulence system random search . . . . .	69
5.3	Statistical analysis of Turbulence system hill climb . . . . .	75
5.4	Statistical significance (t-value) analysis between the performance of Genetic Algorithm, Random Search and Hill Climbing methods. . . .	80
5.5	Target and evolved rule sets for the meta-automaton system . . . . .	84
6.1	Example patterns from the nano system simulator . . . . .	93
6.2	Statistical analysis of Nano system evolutions . . . . .	106
6.3	Extended nanostructure simulator: Parameters . . . . .	112
7.1	Chromosome for the evolution of neural network parameters . . . . .	124
7.2	Statistical analysis of neural network evolutions, showing the resulting parameters as well as the RMS training error . . . . .	125
7.3	Statistical analysis of surrogate-assisted nano system evolutions. The table also shows the evolved genotype, as chosen by the Automated Decision Maker, along with the associated Normalised Combined Fitness value. . . . .	131



## Abstract

The synthesis of abiotic life-like behaviour in complex chemical systems is one of the great scientific challenges in today's research environment. Very often in this type of design, the parameter space is so large and the system so complex that analytical, rational design techniques are extremely difficult to manage, and more often than not, unavailable altogether. Machine learning methods have found many applications in the realm of design and manufacture and the research described in this thesis describes the application of these tools towards the development of pre-specified chemical functionality in complex systems. A detailed description of the 'Evolutionary Engine' built with this sort of design in mind is given, including preliminary investigations into coupling this engine to a 'real life' chemical reactor array. Studies are performed on a range of complex systems, including benchmark problems based on cellular automata, and, for the first time in this domain, on real world problems in self-organised scanning probe microscopy. Given a target behaviour of the system in question, usually represented by a series of patterns in a 2D image, it is shown that parameters can be 'reverse engineered' through a sophisticated combination of machine learning techniques and image analysis methods, such that the target behaviour/pattern can be faithfully reproduced. Finally, techniques for the approximation of a complex system and its associated fitness function are explored, giving rise to a dramatic decrease in computation time with little compromise to the quality of results.

## Acknowledgements

First, my thanks to the Engineering and Physical Sciences Research Council for their funding throughout the three years of my PhD course (EP/D023378/1), and to the Automated Scheduling, Optimisation and Planning research group at the University of Nottingham for the comprehensive training and education they provide.

Particular thanks, of course, go to my supervisor, Dr Natalio Krasnogor for his support, inspiration, good humour and friendship – even when results were discouraging and motivation flagging, our weekly meetings never failed to lift my spirits and to inspire, and I feel privileged to have had the benefit of his care and expertise.

Special thanks also to my dear friends outside of the world of Computer Science, both members past and present of the University of Nottingham Music Society and all those in the Parish of All Saints, St Mary and St Peter, Nottingham – their friendship and all the wonderful music we produced together provided an invaluable counterpoint to scientific research.

To Adele, for her boundless love, encouragement, humour and companionship, my deepest love and thanks, always.

A PhD represents the culmination of over two decades of education – my final thanks, then, go to all the teachers and staff at Bruern Abbey School (particularly Rod Woods, who played such a formative part in my early education); New College School, Oxford; St Edward's School, Oxford and the University of Nottingham, but most of all to two loving parents, who selflessly provided for this most privileged of educations, the appreciation for which is as heartfelt as it is inexpressible.

## CHAPTER 1

# Introduction

The quest to produce ‘artificial life’ has been ubiquitous in the scientific community for centuries. As early as 1738, Jacques Vaucanson’s mechanical duck [101] captured the interest of scientist and layman alike. Of all the research devoted to designing systems such that they exhibit some sort of life-like behaviour, particularly interesting is the work investigating those systems that comprise entirely abiotic components – this aspect of such research breaks with the often firmly-held view that life-like behaviour can emerge only from biological components and, in doing so, opens the field of artificial life research up to a far wider group of researchers across many disciplines.

Of course, such work is entirely dependent on the answer to what ‘life-like behaviour’ actually is. The question of what defines life has provoked countless discussions within the religious, philosophical and scientific spheres. It is important not to assume that all life is like ours, namely RNA/DNA-based biomolecular organisms. Perhaps the biomolecular life that we observe today on Earth is just one of a number of possible implementations! Indeed, it is often argued that life on earth evolved out of a completely different set of building blocks, that were later to be out-competed by ‘modern’ RNA/DNA systems [97]. One aspect of living systems that is surely without controversy is the notion of some container or boundary that separates the system from its surrounding environment, or, in other words, cellularity. Within the confines of this container, occurs a process or processes that result in a particular organisation of components that drives what we deem to be a ‘life-like’ feature, whether this be a method of reproduction, energy exchange/metabolism, information processing, etc..

Another crucial aspect of any of these processes is some sort of ‘self-organisational’ property, and so the study of self-organising systems, from the most basic to the fiercely complex, is absolutely key in this field of research.

During the birth of the science we now know as ‘artificial intelligence’, Alan Turing (often called the ‘father of Computer Science’) proposed a test to answer the question as to when machines could be termed ‘intelligent’ [127]. It is essentially an imitation game: A human judge engages in a series of questions (probing some aspect of intelligence) with two other parties, one a human and the other a machine, each in a separate room. If the judge cannot reliably tell which is which, then the machine is said to pass the test. It is assumed that both the human and the machine try to give the impression of being human. There are a number of objections to this test (perhaps most famously, Searle’s ‘Chinese Room’ concept [111]) and indeed a number of counter-arguments to these objections, but it is still the most widely adopted method of answering the artificial intelligence question. Indeed, we now commonly see the Turing Test at work distinguishing between human and ‘softbot’ in some internet registration pages, where a human user is required to interpret a picture containing an alphanumeric code. These are called CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) [129].

A similar imitation game is proposed in [22] for the recognition of life (or ‘life-like behaviour’), but with a number of necessary modifications. Instead of asking questions to prove the intelligence of a machine, candidate ‘life-like’ systems are probed in a number of different ways, in order to explore a different aspect of life, whether this be response to stimuli, heredity, metabolism, or any of the other commonly used criteria. If these probes cannot distinguish between natural, *in vivo* and artificially constructed, *in vitro* systems, the manufactured system can be reasonably deemed to be ‘life-like’, as far as the probed criterion is concerned.

The aim of the research presented in this thesis is to explore some of the issues involved in the design of complex physio-chemical features that can be seen as initial stepping stones towards this goal of synthesising artificial, complex, life-like chemical behaviour.

Most design and manufacture problems (anything from building a bridge to de-

signing a new computer) are usually solved by rational/analytical design, i.e. by hand, by engineers (if aided by computers), but as the problems that people want to solve get more and more complex, this rational design becomes more and more infeasible, and in many cases there are in fact no analytical solutions available. To use a relevant illustration, consider a simple chemical system. The inputs into this system determine the nature of the output, such as the yield or robustness of the chemical product, and could include parameters such as temperature, concentrations, pH and flow rates. Given, for instance, thirty inputs, each of which can take an integer value in the range  $[0,1000]$ , this results in  $10^{90}$  possible combinations of input values – substantially more than the number of atoms in the universe. Rational design of such a system is, in the majority of interesting cases, infeasible not only because exploring the search space in an exhaustive fashion would take a very long time, but also because the system is such that the mappings between input and output are highly complex, non-linear and counter-intuitive, thus making it possible that no analytical solutions exist. For these reasons, computational methods are often employed to search for the optimum set of input parameters in such cases. Stephen Wolfram describes such methodologies as “a new kind of science” [131].

It is a commonly held view that local search methods are not as appropriate for the optimisation of such systems [87, 102] than those algorithms that stem from the field known as ‘evolutionary computation’. John Holland’s proposal of the genetic algorithm [56] in 1970 was the defining moment for this fascinating computing paradigm which takes the principals of Darwinian evolution and applies them to computational search methods.

Evolutionary computation is not bounded by human expectations or intuitive reasoning; the ‘trajectory’ of an evolutionary design may be highly counter-intuitive, yet highly successful. Some problems may be solvable by rational means only, some only through evolutionary design, some through both, some perhaps by neither. Examples of the success of evolutionary design can be found throughout the literature; particularly noteworthy is the work of Julian Miller with Liquid Crystal Displays [48] where the input voltage patterns needed to create a particular pattern on the LCD display are evolved, and the pioneering studies of Adrian Thompson in the evolutionary de-

sign of analog circuits [122] where solutions were found that were much more efficient than their human-designed equivalents. Many more such examples are referenced in chapter two of this thesis.

This type of automated, ‘machine learning’ assisted design is gaining increased popularity in the world of electronic engineering. Once electronic chip design passed a certain point of miniaturisation, systems began to be embedded into all manner of systems, some critical, such as air craft, weaponry, power generation arrays, and others. As the scale of these systems increased, with millions of transistors connecting to millions of other transistors all on a single chip (so-called, very-large-scale integration (VLSI) circuits), rational design became evermore infeasible. Furthermore, once such a chip was embedded deep into a system, maintenance became very challenging, if not impossible. To address these problems, the notion of “Evolvable Hardware” was introduced [45], whereby circuits could adapt and reconfigure *themselves* in response to changes in their surroundings, such as environmental factors, connection malfunctions, etc.. The ‘Field Programmable Gate Array’ (FPGA) [12] is the most widespread example of this type of device, enabling the programming of its function to be carried out after manufacture, and offering near unlimited reconfiguration opportunities.

## 1.1 Motivation and aim

We suggest that the field of synthetic chemistry has now reached a similar stage to pre-VLSI electronic engineering; although the guiding principals are well-understood, the integration of a large number of complex components means that a more advanced method than rational design needs to be applied. The hypothesis is posited, therefore, that evolutionary algorithms can be employed to coerce complex chemical systems – but also, more generally, self-organised systems – into a pre-determined target behaviour. This thesis therefore explores some of the issues involved in coupling machine learning techniques to chemical design problems in the hopes of contributing to the initial development of the field now emerging as ‘Evolutionary Chemistry’ [54]. It is important that any software developed during this work should be easily available

and easily usable, so that an expert understanding of the algorithms is not required to operate the software, thus enabling its use by researchers in a range of other disciplines. It is hoped that the use of evolutionary computation techniques rather than the more traditional analytical design methods will one day open up a new, previously unreachable area of chemical functionality, whilst also giving insight into the underlying chemistry behind them.

Not only must we specify the algorithms, parameters, etc. that evolve the design for the ‘recipe’ for a given problem, but we must also develop methods of analysing the resultant behaviour and comparing it to that which has been specified by the user. Indeed, verifying whether an evolutionary algorithm is an effective method of optimisation for a given problem using a given fitness function is still an open problem. It is important, therefore, that the complex nature of the genotype–phenotype–fitness relationship is carefully analysed.

One of the most common problems with any sort of experimentation relating to complex systems, particularly simulations, is the often very lengthy run time and computational expense. Approximation models, sometimes called ‘surrogate’ models of complex systems are increasingly well-studied, and so this thesis will also investigate the hypothesis that self-organised complex systems can be modelled by a simpler approximation, resulting in savings of time and computation expense without compromising the quality of results.

To summarise the aim of this thesis into a single problem definition:

### 1.1.1 Problem definition

Complex systems are those that demonstrate a highly non-linear behaviour. They are often multi-dimensional (that is, many input parameters determine their resultant behaviour), noisy (a given parameter set and runtime environment may result in a variety of behaviours) and hard to model. The principal question addressed in this thesis is: how can researchers successfully ‘reverse engineer’ the input parameters to complex systems such that the behaviour – that is, the system’s actions/reactions to the environment – of these systems matches a pre-defined target specification.

To investigate this research question, a number of objectives are defined:

### 1.1.2 Objectives

1. A toolbox of machine learning techniques will be designed, but with the crucial added features of user-friendly configurability, and ‘connectability’ to (remote) laboratory apparatus.
2. A selection of behaviour analysis methods for complex systems will be developed, that can be used to compare designed behaviour with target behaviour in the context of an evolutionary algorithm.
3. Development of a method for verifying the robustness of a given analysis method in the context of complex systems optimisation problems.
4. Approximation models of complex systems will be studied in order to investigate the potential of such models to save time and computational expense without compromising the quality of results.

Self-organisation is a relatively new area of research - there is much still to be discovered, much still to be formalised. A greater understanding of the processes involved in dynamic self-organised systems (examples of which are presented in the next chapter) could herald the start of a revolutionary new paradigm of evolvable chemical complexity. It is towards this overall goal that the research presented within this thesis aims to contribute. Furthermore, it is hoped that this research is presented in an easily readable, accesible manner.

## 1.2 Methodology

1. Careful analysis of the fitness landscape is important in any evolutionary study. Both experimental and statistical techniques and measures are employed to assess the complexity of the genotype–phenotype–fitness mapping in the various domains being investigated.



2. Supported by the landscape analysis, standard evolutionary algorithms are then coupled to the fitness analysis methods being developed.
3. At the initial stage of algorithm selection, the proposed evolutionary computation methods are compared, through benchmarking experiments, to basic standards such as random search and hill climbing methods.
4. The evolutionary algorithms themselves are kept relatively simple. As this is the first time such methods have been applied to, in particular, scanning probe microscopy, over-complication is to be avoided at all costs, and thus we tend to take a ‘bottom up’ approach to algorithmic complexity. Only if the simplest algorithms do not succeed, do we move on to more complex alternatives.
5. Finally, the potential for approximation of the complex system is explored, in the hopes that comparable results may be possible but without the computational expense associated with running complex systems.

## 1.3 Contributions

Research into self-organisation and self-assembly systems is in its infancy, and so it is hoped that the work presented in this thesis will be seen as an interesting contribution to this growing field.

### 1.3.1 Software platform

Although a number of evolutionary algorithm ‘toolboxes’ exist, we believe that the platform developed during this research (and described in chapter 3) is unique through its ease of use, flexibility, configurability and ease of access. Specifically, its intuitive user-interface and ‘connectability’ to laboratory apparatus (such as that described in section 3.3) makes it particularly appealing to our inter-disciplinary colleagues in physics, chemistry and biology. It gives rise to the potential for real-time optimisation of complex chemical processes.

### 1.3.2 Robustness verification

Chapter 4 presents a methodology for verifying the robustness of a given fitness function in the context of a given complex search space. Finding a reliable method of predicting when an evolutionary algorithm will be an effective method of optimisation is still an open topic of research in evolutionary computation theory. Due to the complex nature of the genotype–phenotype–fitness mapping in the problems with which we are dealing, a method of verifying the robustness of a fitness function is of great importance. We present a two stage process for just such a method to verify whether a given fitness function could accurately direct a search in such complex dynamics. This two stage process involves both cluster analysis and fitness-distance correlation, and is published (see section 1.3.5).

### 1.3.3 Evolutionary design of scanning probe microscopy

Chapter 6 applies an evolutionary algorithm to design a specific pre-defined behaviour in a nano-scale self-assembly system. This published work describes a novel application of evolutionary computation methods; the application of such techniques to the domain of scanning probe microscopy is unique. In particular, we believe the use of Minkowski functionals as part of a new graphical similarity metric to be a most interesting innovation. This application takes the work of this thesis much closer to the design of real, physical systems, as well as providing an important link between simulation and experiment in the study of self-organising nanostructured systems. This is novel work and takes us ever closer to the concept of software control of matter. The potential of this work to be extended to the construction of nanoscale components as part of an ‘unconventional computing’ system, such as that suggested in [126] is particularly relevant in today’s research environment, and indeed, there is a clear analogy between the behaviour of such a system and the origins of biological life, though the formation of simple particles into such an arrangement that could encode some form of complex behaviour or functionality.

### 1.3.4 Fitness approximation

Although the results achieved with the methods alluded to above are of an impressive quality, the time taken, both by the simulator itself, but also by the fitness calculation, is not inconsiderable. Fitness approximation is an increasingly well-studied field within the evolutionary computation community, and explores the potential for savings in time and computational expense through the use of a fitness approximation model. It is shown in chapter 7 that a neural network ensemble can accurately embody not only the behaviour of the complex system, but also its subsequent mapping onto the relevant analysis methods. This use of a single model to approximate both the complex system *and* the analysis methods is an interesting and ambitious contribution to this area of study.

### 1.3.5 Publications

The work towards – and contained within – this thesis has resulted in the following peer-reviewed publications, posters and workshop presentations:

1. L Li, **P Siepmann**, J Smaldon, G Terrazas, N Krasnogor: Automated Self-Assembling Programming. *Systems Self-Assembly: Multi-Disciplinary Snapshots*, Elsevier, pp. 281-303, 2008.
2. **P Siepmann**, CP Martin, I Vancea, PJ Moriarty, N Krasnogor: A genetic algorithm approach to probing the evolution of self-organised nanostructured systems. *Nano Letters*, 7(7): pp. 1985-1990, 2007. [Impact factor: 10.317]
3. G Terrazas, **P Siepmann**, N Krasnogor, G Kendall: An evolutionary methodology for the automated design of cellular automaton-based complex systems. *Journal of Cellular Automata*, 2(1): pp. 77-102, 2007. [Impact factor: 0.684]
4. L Cronin, N Krasnogor, BG Davis, C Alexander, N Robertson, JHG Steinke, SLM Schroeder, AN Khlobystov, G Cooper, P Gardner, **P Siepmann**: Is it alive? Recognising Cellular Systems: A Computational-Chemical Perspective. *Nature: Biotechnology*, 24, pp. 1203-1206, 2006. [Impact factor: 22.30]

5. **P Siepmann**, G Terrazas, N Krasnogor: Evolutionary design for the behaviour of cellular automaton-based complex systems. In the *Proceedings of the Seventh International Conference of Adaptive Computing in Design and Manufacture*, pp. 199-208, 2006.
6. **P Siepmann**, CP Martin, N Krasnogor, P Moriarty: A genetic algorithm approach to guiding the evolution of self-organised nanostructured systems. Conference presentation at *Condensed Matter and Materials Physics*, Exeter, UK, April 2006.
7. L Bianco, D Pescini, **P Siepmann**, N Krasnogor, FJ Romero-Campero, M Gheorghe: Towards a P-systems *Pseudomonas* quorum sensing model. *Membrane Computing*, Springer, pp. 197-214, 2006.
8. **P Siepmann**, N Krasnogor: A Java RMI-based application for remote automatic job submission. Poster at the *Nottingham High Performance Computing Workshop*, Nottingham, UK, January 2007.
9. **P Siepmann**, CP Martin, N Krasnogor, P Moriarty: A genetic algorithm approach to guiding the evolution of self-organised nanostructured systems. Workshop presentation at *Embodied Evolution of Complex Experimental Systems*, European Centre for Living Technology, Venice, Italy, May 2007.

## 1.4 Overview and structure

The next chapter ‘sets the scene’ for the themes of this thesis, placing them in the context of a survey of relevant literature. Having presented an overview of evolutionary computation, and genetic algorithms in particular, the ‘Evolutionary Engine’ software system is then presented in chapter 3, including details of its ability to connect to a large bank of parallel computation nodes and/or a remote chemical reactor array. A comprehensive robustness verification protocol for evolving complex systems is given in chapter 4. Chapter 5 presents the first significant results – a study of a cellular automaton-based system that simulates the behaviour of fluid flowing through

a pipe; it is shown that given a graphical representation of some desired resultant behaviour, input parameters to the system can be ‘reversed engineered’ so as to reproduce the target behaviour. In chapter 6, similar strategies are then applied to a Monte Carlo simulation of a well-characterised physical process – the self-organisation of a particular nanoscale system. Not only is it shown that parameters can be reversed engineered to match a wide variety of self-organisation patterns, but, in chapter seven, that the entire system can be surprisingly well captured by a simple ensemble of neural networks.

## CHAPTER 2

# Context and background - a literature review

Relevant background and reference to previous work related specifically to the individual topics covered in this thesis is given when the subjects are introduced. This section aims to ‘set the scene’ for what follows, laying out the theme of machine learning applied to complex system design in the context of artificial life research and the surrounding literature.

Any research towards life-like behaviour in any context requires a definition of life itself. A clear and agreed definition of life is arguably one of the most elusive concepts in the scientific (indeed, philosophical, religious and linguistic) communities. Indeed, Palyi et al. [93] went as far as to suggest that life “is what the scientific establishment will accept as life”! The authors of [91] provide a useful overview of this vast minefield of a topic. “Life is like music; you can describe it but not define it”, says Lazcano in [70] - that is, we find it much easier to recognise life than to define it. But even the recognition of life needs precise descriptors if we are to provide a scientific proof of a system behaving in a ‘life like’ manner. Perry and Kolb [95] make the observation – particularly relevant in our case – that the point at which chemical non-life becomes biological life is particularly hard to identify. Oliver and Perry’s [91] preferred definition is that “life is the sum total of events which allows an autonomous system to respond to external and internal changes and to renew itself from within in such a way as to promote its own continuation”. In the quest to design ‘life-like behaviour’, though, it is important to note that the aim is not this ‘sum total’, but

rather its component parts, and how to define these is equally insidious. As Oliver and Perry suggest, what is really needed is not a single, grand definition, but rather a series of working descriptions. The authors of [50] and us in [22] take this a step further in suggesting the Turing Test equivalent discussed in the previous chapter.

## 2.1 Self-organisation

One aspect of these arguments that can be widely agreed upon is that the process of self-organisation is a fundamental pre-requisite to life. All systems (organisms) we consider to be alive demonstrate some form of self-organisation, some more finely grained (such as the formation of cell walls) than others (such as the flocking behaviour of birds). Self-organisation is the process by which a collection of autonomous components (of whatever form), arrange themselves into aggregated structures. There is no ‘global control’ or ‘master plan’ – the organisation process is driven entirely by the local interactions between components themselves, and with their local environment. As observed in [117], such systems will often demonstrate so-called ‘emergent properties’, that is to say, behavioural properties or functions that are not evident in the individual components that comprise the whole. This non-linearity of such systems mean that their design, in particular, is a notoriously difficult problem, and one of the aims of this thesis is to illustrate some examples of how such non-linear, complex systems can be coerced into performing some pre-defined, target function.

Self-assembly is a slightly different, yet related process, whereby the self-organising components converge to a stable, equilibrium state (in a self-*organised* environment, the system is out (or sometimes kept out) of equilibrium). In the most interesting cases, these self-assembled structures will have a very definite function encoded within its ordered state. If these natural processes could be better understood, and their power harnessed for the production of man-made components, it would herald a revolution in the production of embedded control systems, smart drug systems and robotics, to name just a few. Through an ability to coerce self-organised systems to behaviour in a certain way, a much wider variety of substances could be “functionalised” to perform a given task. Moreover, such systems can, as observed in [8]

exhibit properties of adaptability, self-healing and self-replication.

It is crucial to appreciate at this point, that even the simplest of self-organised systems can perform useful life functions, which can, through Rothemund's observation [105] that "self-assembly and computation are linked by the study of mathematical tiling", also be considered to be 'natural' computations.

The work of Paun and Gheorghe (as described in [40]), demonstrates how computational processes can be modelled by self-organising systems; in [13, 28], we see how simple tile systems (of which DNA can be seen to be one) can be interpreted to carry out arithmetic operations, and in [10] how self-assembly processes can even be interpreted as their own class of programming language. A particularly comprehensive study of the design of tile systems can be found in [118]. Moreover, the fascinating work of Winfree and Schulman [110] show how tile systems can be programmed in specially designed DNA crystals, giving an important insight into how such systems might have developed *in vivo*.

The authors of [71] give an interesting demonstration (with computational components as exemplars) of how such systems can develop into larger, more complex systems by the application of the process of evolution. This relationship between self-assembly/self-organisation and evolution provides the principal motivation for the research contained within this thesis; the formation of complex self-assembled systems driven by the process of competitive selection is a vital component in the understanding of the origins of (artificial or otherwise) life. In applying evolutionary computation techniques to life-like chemical components, one comes startlingly close to meeting NASA's accepted definition that "life is a self-sustained chemical system capable of Darwinian Evolution".

Driving self-organised systems into a particular behaviour is an intrinsically difficult problem; the smaller and simpler the nature of the individual components of the system, then so the system's "intelligence" gets split into smaller and simpler units, making the process of programming or designing particular behaviours considerably more challenging, due to the system's increased level of distribution. One of the principal aims of this thesis is to address this problem, looking at a number of examples of self-organised systems and the methods that can be applied in order to achieve



designed complex behaviour.

## 2.2 Complex system design

In [44], the authors reflect that “perhaps the greatest concern is how do we build artificial systems (or manage natural ones) so that the properties that emerge are the ones we want”. It is important to note that in this thesis, we are concerned principally with this evolution of complexity – though there is a great deal of work on a plethora of meta-heuristics designed for large, complex and noisy search spaces [55, 96, 85, 67, 68], this is not the focus of this work; rather this thesis aims to address some of the problems involved in interpreting complex behaviour such that it is suitable for optimisation *by whatever method*. As the mainstay of evolutionary computation, and one of the most widely used global optimisation technique, the majority of research presented throughout this thesis makes use of a standard genetic algorithm. It is anticipated that future research directions would include the application of our methods to other related problems.

It is long established that local search methods are not appropriate for the optimisation of such complex problems, whose search spaces can be expected to be highly non-linear and multimodal. In particular, [87] and [102] support the reasoning that for optimisation of this type, a global metaheuristic has a much greater chance of success. One of the most widely used families of global metaheuristic are those that fall into the category of evolutionary computation, and as we have already discussed, similar methods have already proved revolutionary in the field of electronic engineering. John Holland’s proposal of the genetic algorithm [56] in 1970 was the defining moment for this fascinating new computing paradigm. If, as Charles Darwin proposes, natural selection can function over natural systems to produce complex organisms, then it is not a giant leap to apply evolutionary computation to self-organising processes in an attempt to produce designed complex systems, whether this be in the context of a computational simulation, or *in vitro* through a computer controllable chemical reactor array.

## 2.3 Evolutionary Computation

The theory of evolution is now widely understood and accepted, and no better explanation of the process can be found than the original works by Charles Darwin [23, 24], though the subsequent thinking of Richard Dawkins [25, 26, 27] is invariably insightful, thought-provoking, and often entertaining (even if the effective communication of the science is too often obstructed by the author's militant atheism). The field of evolutionary computation draws its inspiration from the natural process identified by Darwin where, in a population of breedable members with heritable characteristics, set in an environment with limited resources, well-performing members of the population have a greater chance of surviving and procreating than less well-performing members; thus emerges the process of evolution where successive generations inherit the 'good' features of the previous generation but not the 'bad' ones (in the ideal case). An excellent introduction to the computation applications of these theorems is given in [35], whilst [71] presents a fascinating view of how complex features are derived through the process of evolution, using 'digital organisms' as exemplars.

Just as in Darwinian evolution where there is a population of individuals, each with their own genetic code ('genotype') that determines their physical, chemical and biological characteristics (referred to as the individual's 'phenotype'), a genetic algorithm (GA) maintains a population of individuals where each individual encodes a possible solution to a problem. Each solution is evaluated and assigned a 'fitness' value. This evaluation is performed by passing the solution into an objective function. This function could be a chemical reaction, a model, a simulation, a game or whatever is appropriate to the problem under consideration. The output of the function could then be a colour value, a distance, a score or whatever performance assessment is appropriate to the problem under consideration.

For example, we could investigate the famous Travelling Salesman Problem [64] by initialising a population of individuals where each genotype encoded the ordered list of cities; in GA terminology, each city label is a 'gene' and the entire list is the 'chromosome' which defines the individual's 'genotype'. The 'phenotype' is the TSP tour itself, and the objective function calculates the fitness value as the total

distance travelled. In this problem, the lower this value, the better a solution is considered (a so-called ‘minimisation problem’). In other types of problem, such as many of those considered in this thesis, the genotype–phenotype–fitness mapping can be significantly more complex. Although GAs were originally conceived with purely binary genotypes in mind, much of the subsequent research in the area has developed the concept into real-valued encodings, and even multiple datatype chromosomes.

Well-performing solutions/individuals are then permitted to ‘breed’. In the best case, a given child will inherit good genes (parameters) from each parent and thus will be expected to perform as well or better. In this way, each successive generation demonstrates a higher average performance (fitness).

Genetic algorithms are useful because they allow us to sample a search space in an intelligent manner. Instead of exhaustively testing every possible genotype (combination of input parameters) which may take an infeasibly long time, a genetic algorithm can take a sample from across the search space and quickly ‘zoom in’ to the regions of the space where good solutions appear to reside. In the best case, the optimum solution will be found.

Let us now look at this process of ‘breeding’ in a little more detail. The population is initialised with a number of (usually randomly created) individuals, although the population could be seeded with a number of individuals known, for instance, to have a high fitness. The genetic activity then begins, as illustrated in algorithm 1.

### 2.3.1 Selection

In each iteration, two individuals are selected to be parents. Usually the selection probability is proportional in some way to their fitness. The EE supports two of the most popular selection operators [41]: **Roulette wheel selection** essentially assigns each individual a ‘slice’ of a virtual roulette wheel whose size is proportional to the individual’s fitness. Thus, fitter parents have a greater probability of being selected when the wheel is virtually spun, but all individuals in the population have *some* chance of selection. **Tournament selection** selects  $n$  members of the population at random and selects the fittest of this selection to be a parent. The same method

```

while stopping condition not fulfilled do
    parents = select parents from population;
    if random number > a defined parameter then
        | 'mate' the parents to form (usually) two children;
    else
        | children = parents;
    end
    if random number > a defined probability then
        | mutate children;
    end
    insert children into population;
end

```

**Algorithm 1:** Pseudocode to show the standard activity of a genetic algorithm

would be used for the second parent. The parameter  $n$  is user-specified.

The choice of selection operator drives the ‘selection pressure’ of the evolution. It is important that this pressure strikes an effective balance between driving the algorithm towards better and better solutions, and maintaining a reasonable level of diversity amongst the population. Too high a selection pressure (for example, always choosing the fittest two members of the population) may result in a substantially lessened diversity, and thus a premature convergence to a suboptimal solution. In [41], Goldberg and Deb present a comprehensive analysis of various possible methods for parent selection.

The two parents selected (by whichever method) then combine to create a number (usually two) of children where each child individual inherits some genes from each parent. There are a number of different operators available for this ‘crossover’ operation:

### 2.3.2 Crossover

Crossover is the method by which new individuals are formed. Three of the most popular operators are implemented here: **Uniform crossover** [116] is the simplest of the operators. For each gene, the algorithm determines randomly from which

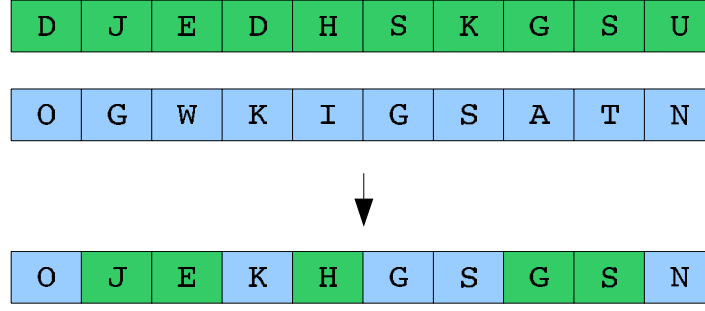


FIGURE 2.1: Graphical representation of the uniform crossover method of parent selection in a genetic algorithm

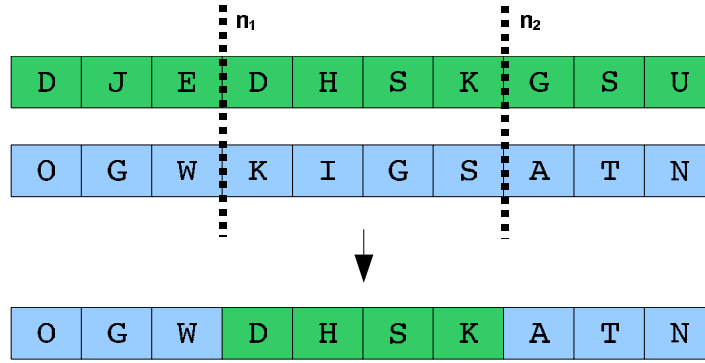


FIGURE 2.2: Graphical representation of the n-point crossover method of parent selection in a genetic algorithm

parent to take the data. Usually each parent has a 0.5 probability of being chosen. See figure 2.1.

Often there will be interdependencies between genes in the chromosomes, i.e. a particular gene might only result in a high fitness if its neighbouring gene has a particular value. For this reason, it is sometimes beneficial to keep groups of genes together during the crossover process. This can be accomplished using **n-point crossover** [29]; given a chromosome of length  $k$ , the chromosomes of both parents and child are split into  $n$  sections, where each child section is taken from a different parent. The parameter  $n$  is user-defined (see figure 2.2). Some implementations of genetic algorithms support mixed datatype chromosomes (that is, genes can take either integer, binary or floating point values) – it is important, therefore, that this datatype information is preserved from parent to child.

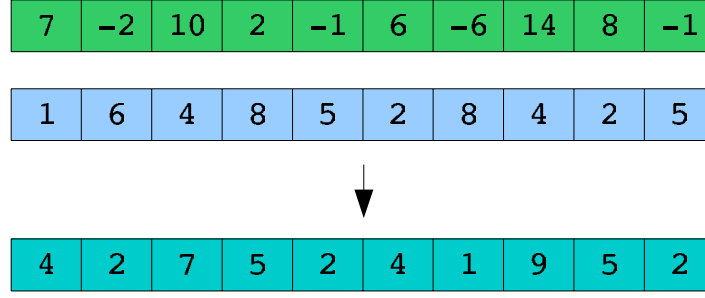


FIGURE 2.3: Graphical representation of the uniform arithmetic crossover method of parent selection in a genetic algorithm

For real valued chromosomes (or parts thereof), the **uniform arithmetic crossover** [83] operator simply sets the child gene to be the arithmetic mean value (or sometimes a weighted mean) of the two parent values. See figure 2.3.

### 2.3.3 Mutation

After recombination, just as in nature, a mutation can occur with some small probability. This can have the effect of taking the search into a perhaps previously unexplored or unreachable area of the parameter space. The **flip** operator is applicable only to binary typed genes and simply reverses the bit from 0 to 1 or 1 to 0. The extension of this operator for integer types is the **random reset** operator where the gene is set to some random value within the bounds of that gene (the lower and upper bound of each gene are problem-specific, and thus user-defined). If an accuracy value has been specified for floating point typed genes (thus making discretisation possible), this operator can also be used for this type of gene. The popular **Gaussian** [52] operator adds to the gene a value chosen randomly from a Gaussian distribution with mean zero and a user-specified standard deviation (by default, this is set at 10% of the entire range of the gene in question, as proposed in [53]). As is evident from figure 2.4, the majority of the Gaussian distribution is close to the original value, meaning large mutations are significantly less probable than small ones.

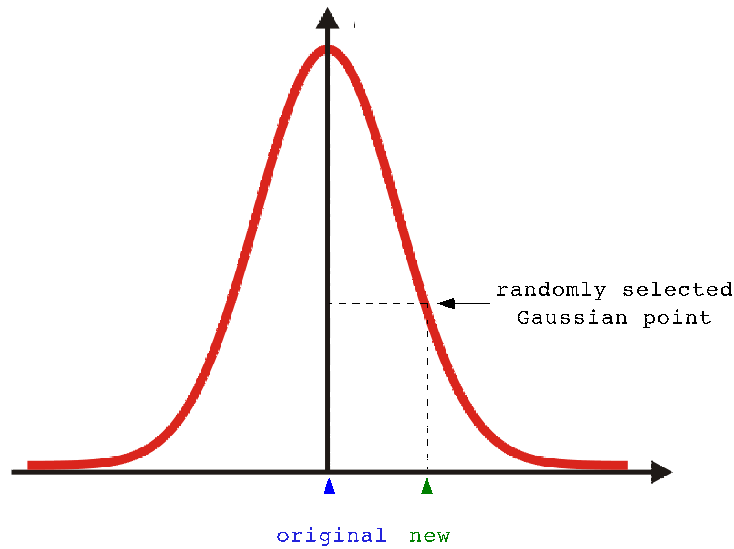


FIGURE 2.4: Graphical representation of a Gaussian mutation operator

#### 2.3.4 Replacement

The children are then inserted into the population, usually replacing less fit individuals.  $\lambda$  is often used to represent the number of children produced and  $\mu$  the size of the population. Users have the choice of either a  $(\mu + \lambda)$  replacement strategy, where the children and parents are considered together and the best (fittest)  $\mu$  individuals are chosen to form the next generation's population, or the  $(\mu, \lambda)$  strategy where only the children are considered. GA implementations often implement an **elitism** mechanism, where a user-specified number of the fittest individuals from the parent population are guaranteed to be passed on to the next generation. This option is often used when a 'generational' approach is taken to replacement, that is, all the parents are totally replaced by their children at every generation. See figure 2.5.

#### 2.3.5 Multi-objective optimisation

Many real-world problems cannot be easily mapped onto a standard genetic algorithm, where each individual is assigned a single fitness value. Many problems will have several objectives that need to be optimised [4, 92], where some may be min-

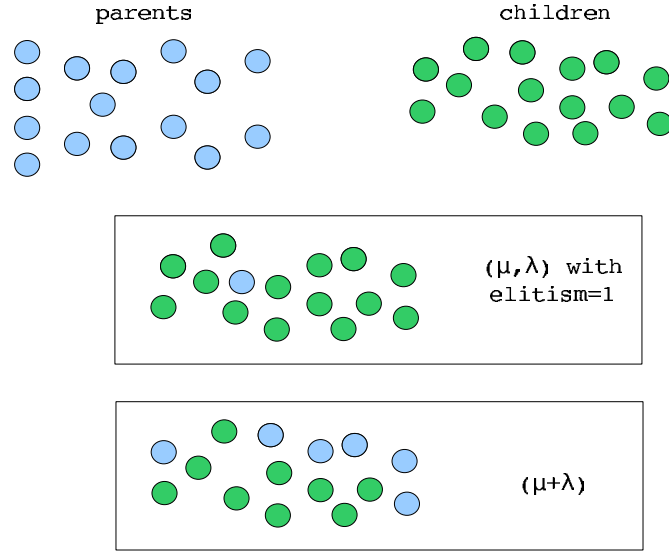


FIGURE 2.5: Graphical representation of replacement strategies available for use within a genetic algorithm

imisation problems and some maximisation problems. Furthermore, some or all of the objectives may conflict with one another. This need for the optimisation of multiple objectives gave rise to the multi-objective genetic algorithm (MOGA). An excellent historical guide to this field is given in [20].

As shown in figure 2.6, a MOGA simultaneously optimises each objective independently of the others, ultimately presenting the user with a set of solutions that satisfy each objective to varying degrees. This set is called the Pareto set (after Vilfredo Pareto) [21] and consists of so-called ‘non-dominated’ solutions. A solution is non-dominated if every objective is the same or better than those of the other solutions in the set, and at least one is strictly better:

$$\text{A solution vector } A \text{ dominates } B \text{ iff } \forall a \in A \wedge \forall b \in B, a \geq b \wedge \exists a \text{ such that } a > b \quad (2.1)$$

The user can then choose which solution to accept, i.e., to which objective to give preference, or an automatic decision maker can be defined appropriate to the problem. The diversity of the Pareto front is an important aspect of a MOGA’s success. To be a satisfactory method, the user should have a wide choice of points (each of which, as discussed above, satisfy each objective to a different degree). This diversity is often



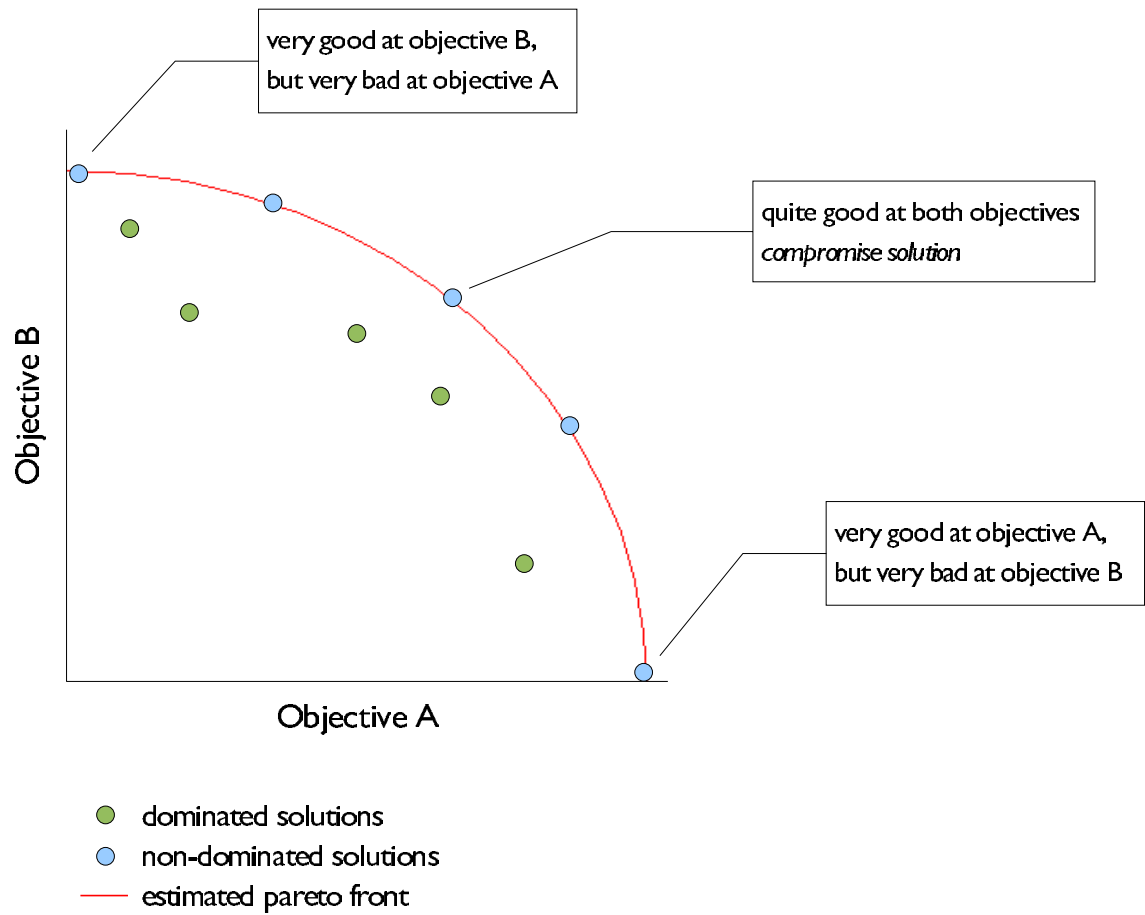


FIGURE 2.6: A graphical explanation of Pareto optimality

measured (and indeed, maintained) through use of the measure known as ‘crowding distance’, defined for each point simply as the sum of the Euclidean distances in each dimension to the nearest neighbouring point.

One of the most successful MOGA implementations is the Non-dominated Sorting Genetic Algorithm (NSGA-II) invented by Deb [31] and widely used in industry.

## 2.4 Frameworks

A number of ‘frameworks’ or ‘toolboxes’ of Evolutionary Computation (EC) techniques have been assembled in the hopes that the use of such technologies might become more widespread in the scientific community. Many such packages, such as EO [65] and ECJ [76] include extensive code libraries, comprising large numbers of components that can be used to construct an EC algorithm for the user’s given problem. Packages such as these, and the inclusion of EC components into widely used analysis tools such as ‘Matlab’ have greatly helped the progression of EC further into the practical scientific community, but the advanced knowledge of computer programming needed to use many of these library-type toolboxes can be prohibitive.

However, the development of systems that make EC-based methods available to end-users in a user-friendly, easily configurable, non-expert fashion is starting to take off. The ‘GUIDE’ system is one such piece of software, which allows the algorithmic components to be specified using an intuitive graphical interface. As explained in [109], although the specification of the evolutionary algorithm can be performed in this high level manner, there are also a number of ‘problem specific’ features (such as the nature of the genome, the definition of ‘fitness’, etc.) where a lower level style of specification is unavoidable.

All the packages mentioned above are software systems that must be downloaded and run on the user’s machine. Surprisingly, in this internet-driven age, examples of remote, web-based EC systems are sparse indeed. The ‘Evolutionary Engine’ presented later on in this thesis aims to fill this ‘gap in the market’, presenting a user-friendly, easily configurable, web-based system in which the evolutionary algorithms run server-side, displaying results via a standard web page.

## 2.5 Applications

Evolutionary computation, and genetic algorithms in particular have earned great popularity both in the theoretical domain [64, 57, 58, 119] and in a great many practical applications; some of the most fascinating deal with the open-ended evolution of structures that must satisfy a number of (often engineering or architectural) constraints. In these cases, the evolution process often produces novel designs that are not considered by rational or analytical methods [99, 100]. Particularly relevant to this discussion are those applications in the design and manufacture of complex engineering problems such as bridge [37] and dome [113] design through to the evolutionary design of circuitry [49, 48, 122], aircraft [90], spacecraft [75] and cellular automata [86, 108, 16, 107, 114].

This last class of problem – cellular automata – are of particular interest as they have been used extensively to simulate complex systems in chemistry [130, 66], physics [124, 19], and numerous other fields of research. Indeed, there is a growing body of work concerned with applying evolutionary computation techniques to the design of cellular automaton-based systems [107, 3, 11], tile systems [119, 46] and P-system models [103, 104].

Crossing the ‘reality gap’ [121] from simulation of chemical processes ([32] presents a particularly extensive review of this field) into *in vitro* optimisation, i.e., the coupling of evolutionary computation methods with real, physical reactor arrays, is still an area of research very much in its infancy. The Belousov-Zhabotinsky reaction has received particular attention [77], due in particular to its potential for unconventional computing [43] through the construction of chemical logic gates [115] by oscillating pattern formation [133]. Adamatzky *et al.* show a particularly intriguing control system for the optimisation of this class of chemical behaviour [15].

Indeed, this field of unconventional computing is a particularly practical application of the type of complex system design described in this thesis. This area of the computer sciences tries to perform computation using techniques other than the standard von Neumann architecture and transistor/integrated circuit technologies. One of the driving motivations behind this field of research is Sir Roger Penrose’s sugges-

tion [94] that human thought is not a Turing algorithmic process, and thus if we are to produce life-like thought processes, a conventional Turing machine-style computer might not be an appropriate method. Furthermore, there are limits to the media in which conventional computers can be embedded – the use of ‘unconventional computation’ methods would enable a greater range of materials to be ‘functionalised’. As well as techniques such as optical computing [42] and quantum computing [33], methods of DNA computing and chemical computing [15] are particularly relevant in the context of this research.

## 2.6 Conclusions

It is alongside work such as that discussed above that the research presented in this dissertation sits. As well as expanding on some of the work related to the evolutionary design of complex systems in general, not least through the suggestion of a method for verifying the robustness of a particular objective function [120], one of the principal areas addressed by this thesis is the evolution of some pre-defined target behaviour in a particular class of self-organised nanoscale system [81]. The application of evolutionary computation to this problem domain is novel, and one of the principal contributions of this thesis. The potential of this work to be extended to the construction of nanoscale components as part of an ‘unconventional computing’ system, such as that suggested in [126] is particularly relevant in today’s research environment.

## CHAPTER 3

# The Evolutionary Engine

The hypothesis posed at the outset of this thesis surmised that evolutionary algorithms can be employed to coerce self-organised systems into a pre-determined target behaviour. To investigate this claim, it is necessary to develop a platform on which to perform such evolutionary computation. This chapter describes the ‘Evolutionary Engine’ built for this purpose.

Genetic algorithms are the mainstay of evolutionary computation and one of the most powerful and widely used methods in the machine learning toolbox. They are particularly suited to optimisation problems involving large search spaces and/or complex fitness functions. It is a matter of debate why evolutionary computation methods are still not regarded as mainstream methods in fields such as engineering, chemistry and physics, but one common complaint by potential end-users in the scientific community is the lack of a user-friendly framework; although many algorithms have freely available source code, setting them up, particularly in a “wet” laboratory context, requires considerable knowledge and expertise. If the computational support package proposed in this thesis is to be taken up by our laboratory-based collaborators, a user-friendly, yet powerful and flexible interface is of paramount importance. The ‘Evolutionary Engine’ described in this section aims to be such an interface. It is accessible via the Internet (<http://huey.cs.nott.ac.uk/chellware>), and provides the ability to customise functions and operators, and set options in a user-friendly environment.

The Evolutionary Engine (EE) has been specially designed for optimising design and manufacturing problems. It is a web-based system, implemented using the

Java programming language, coupled with the Tomcat server technology. It is vital that the engine can be tailored to solve a broad range of problems, hence very few options/parameters are ‘hard coded’ into the system, but rather these are user-customisable through a web-based configuration facility.

### 3.1 System architecture

The number and data type of genes in the chromosome, along with parameters for the GA (including the user’s choice of selection, replacement, recombination, mutation operators and rates, as well as the number of objectives to be optimised) can be specified in the configuration module (a ‘screenshot’ of which is shown in figure 3.4), which builds an XML script such as that shown in figure 3.1. Note in the example script shown, each gene has a unique datatype-range combination, hence each is defined within its own ‘block’ (within which crossover takes place). Recombination and mutation operators are specified separately for each block. Note also the definition of the fitness module (‘Nano’) along with the filename specifying the desired target behaviour. This is a single objective problem, hence only one objective tag is present, specifying the bounds and direction of search.

This script, along with a problem specification class (which, most importantly, defines the fitness function), configures the GA to the specific problem to be optimised. This class is user defined, through the extension of a Java abstract class. All problem specification modules must extend the `Problem` class which defines the required methods (illustrated in figure 3.2).

The execution of the GA can then be started and observed over the Internet through a Java servlet, as shown in figure 3.5. This system enables a number of users to run tailored instances of the GA in parallel on different problems.

The customisation of data types is made possible by Java’s powerful type polymorphism [18]. For example, the data encoded into a gene is stored as type `Object`. The XML configuration script can then specify the data type into which this data is cast. A fuller illustration of this application of polymorphism, as well as the class structure of the engine is shown in figure 3.6.

```

1  <chellware user="pas">
2      <chromosome recombination-rate="0.9">
3          <block id="0" recombination-type="uniform crossover"
4              mutation-type="random reset" mutation-rate="0.1">
5              <gene name="MR" type="int" lower="5" upper="50"/>
6          </block>
7          <block id="1" recombination-type="uniform crossover"
8              mutation-type="Gaussian" mutation-rate="0.1">
9              <gene name="coverage" type="double" lower="0.05"
10                 upper="0.49" step="0.01"/>
11          </block>
12          <block id="2" recombination-type="uniform crossover"
13              mutation-type="Gaussian" mutation-rate="0.1">
14              <gene name="kT" type="double" lower="0.5"
15                 upper="2" step="0.1"/>
16          </block>
17          <block id="3" recombination-type="uniform crossover"
18              mutation-type="Gaussian" mutation-rate="0.1">
19              <gene name="EL" type="double" lower="1"
20                 upper="4" step="0.1"/>
21          </block>
22      </chromosome>
23      <fitness module="Nano" target="labyrinth.png.thres.png">
24          <objective type="double" lower="0" upper="max"
25              direction="minimising" plot="on"/>
26      </fitness>
27      <population size="50" selection="roulette wheel" offspring="25"
28          replacement="plus" elitism="0" seed="0"/>
29      <distribution cluster="yes"/>
30  </chellware>

```

FIGURE 3.1: An example XML configuration script used within the Evolutionary Engine. Parameters governing aspects of the genetic activity are shown in green, and those relevant to the specification of the problem itself in red. XML tag names are shown in purple.

```
abstract class Problem {  
  
    // retrieves, for example, the target image  
    public abstract Object getData();  
  
    // run on termination of the GA  
    public abstract void setdown() throws Exception;  
  
    // run on initialisation of the GA  
    public abstract void setup(GA g) throws Exception;  
  
    // stopping condition  
    public abstract boolean stop();  
  
    // evaluates a set of individuals  
    public abstract Object[][] evaluate(Individual[] inds, boolean cluster);  
  
}
```

FIGURE 3.2: Code outline of the Evolutionary Engine's problem specification class.



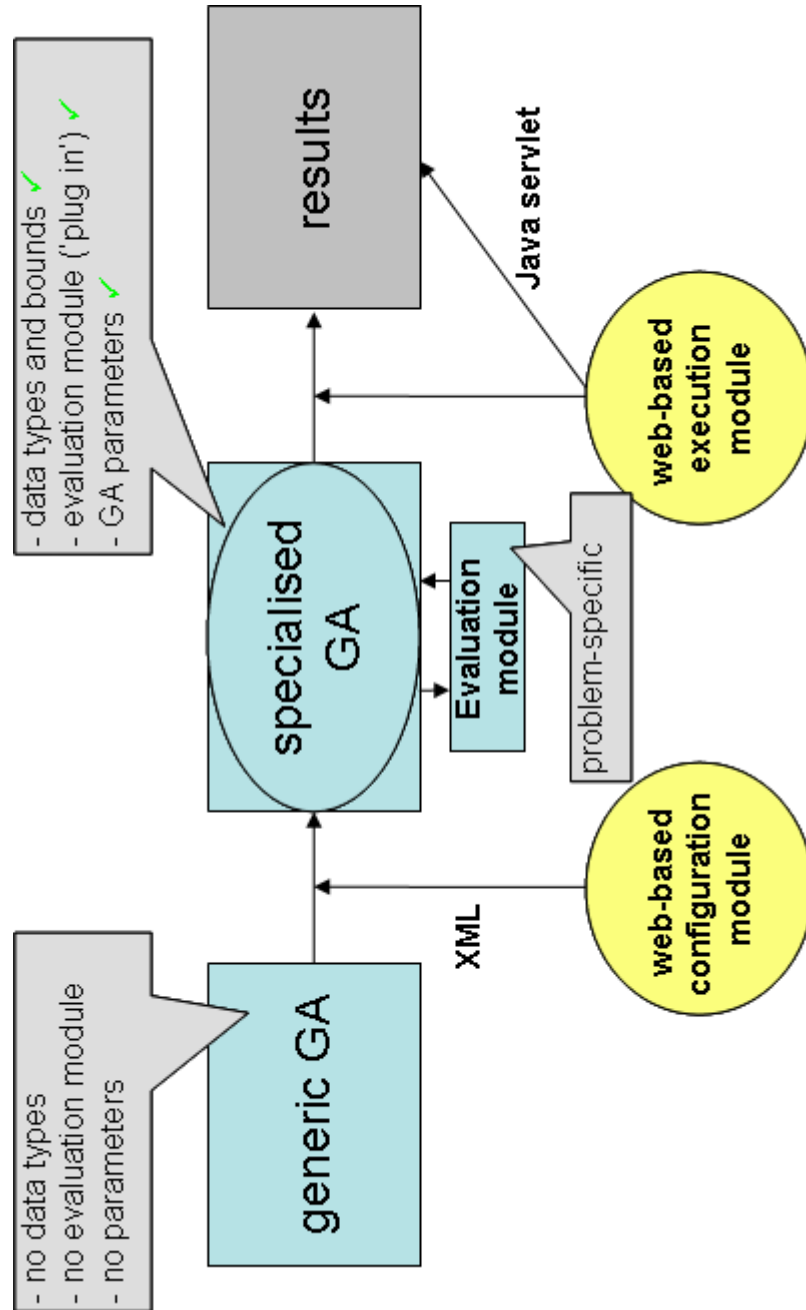


FIGURE 3.3: Diagrammatic representation of the logical structure and operation of the Evolutionary Engine

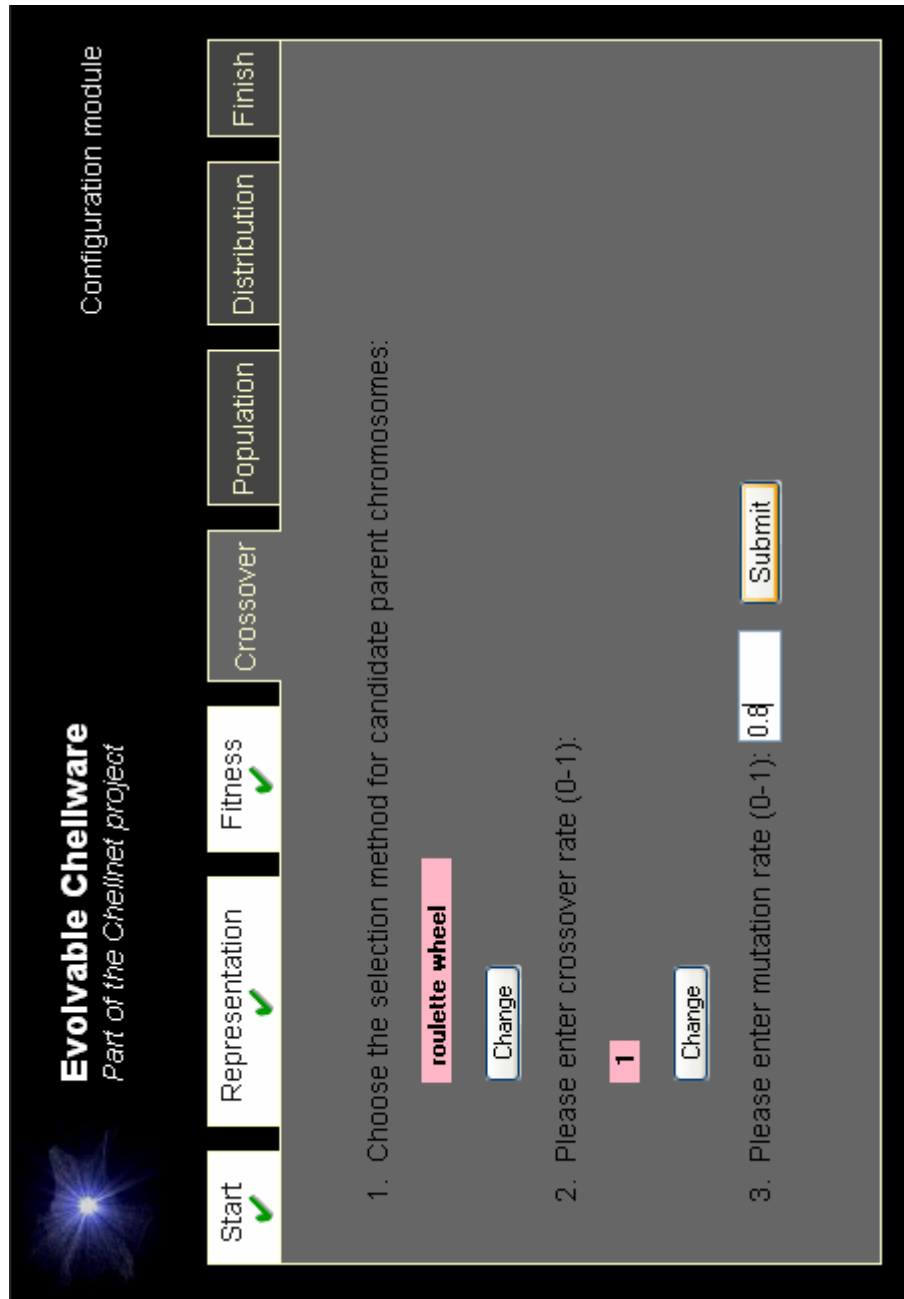


FIGURE 3.4: Screenshot of the web-based configuration module within the Evolutionary Engine

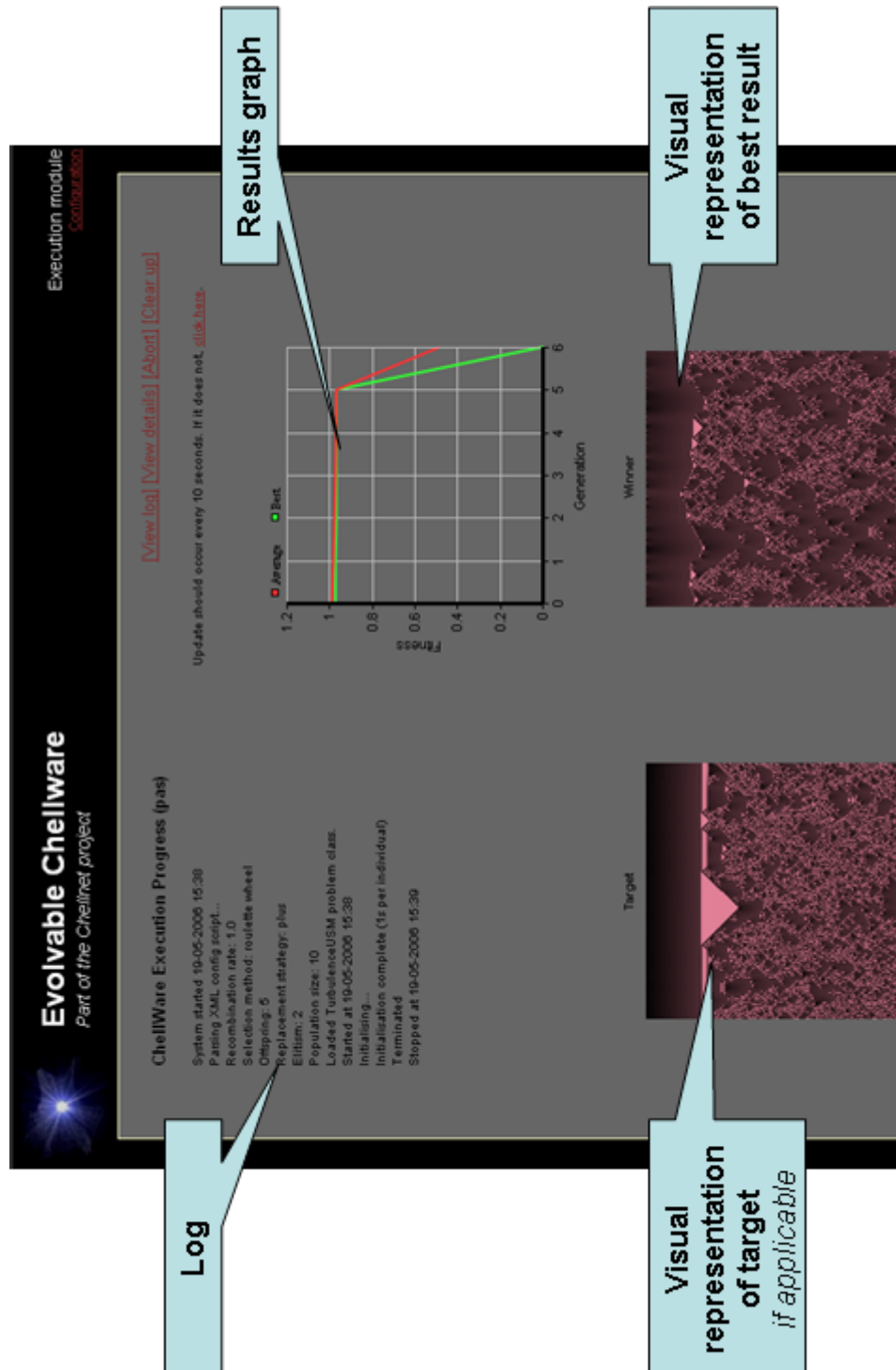


FIGURE 3.5: Screenshot of the web-based execution module within in the Evolutionary Engine

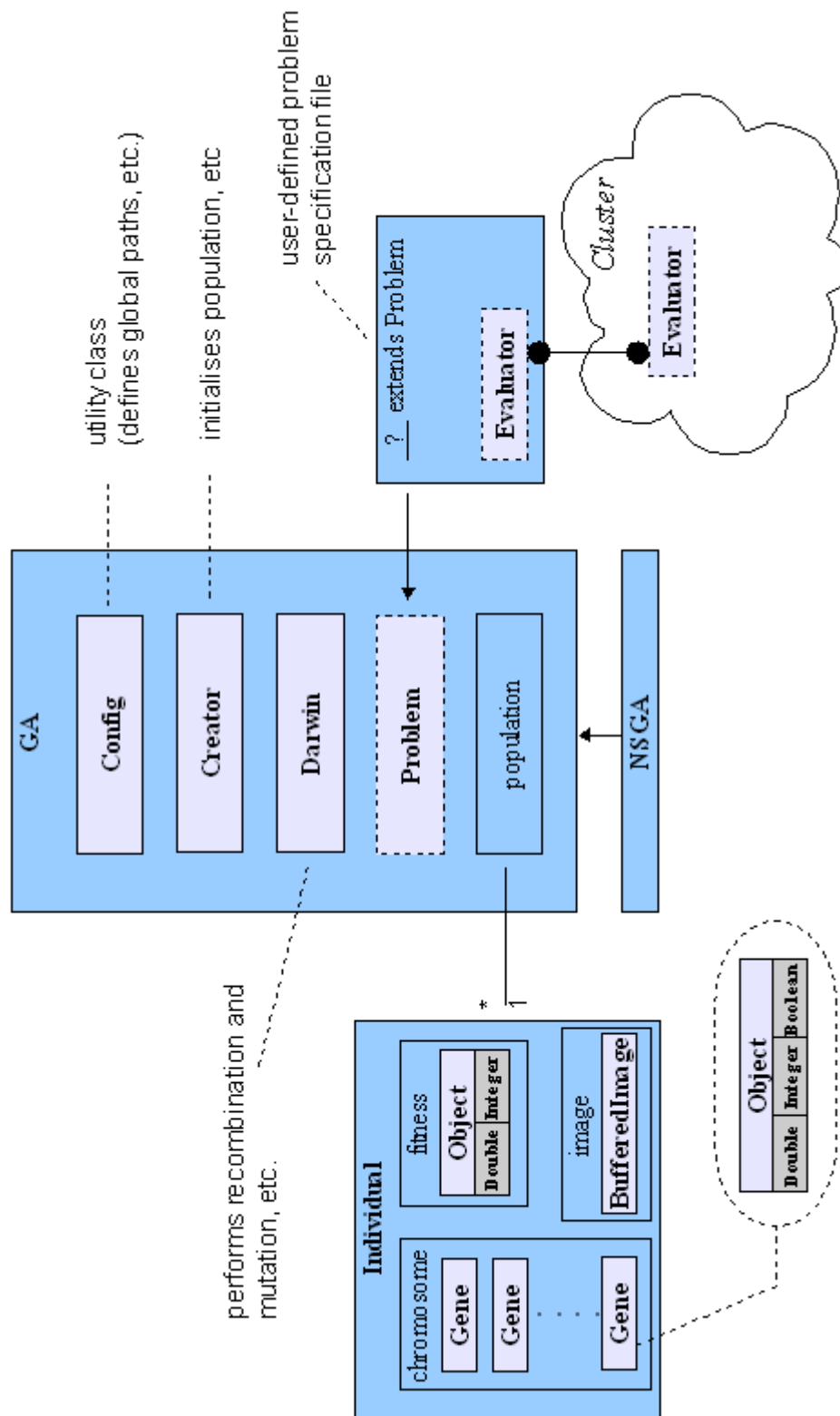


FIGURE 3.6: Diagrammatic representation of the Evolutionary Engine's class structure

The engine supports a number of different options, operators and algorithms, as discussed above and shown in table 3.1.

### 3.2 Parellisation

Many of the fitness functions run on the engine are computationally intensive. Therefore, the engine makes use of the University of Nottingham’s high performance computing facility (or ‘cluster’), enabling the simultaneous execution of up to 1024 processes. Communication between the evolutionary engine and the cluster’s head node is through a specially designed RMI interface [34] as shown in figure 3.7. RMI (Remote Method Invocation) is a programming paradigm whereby high-level constructs, such as method calls, can be implemented across network boundaries. Java has a particularly good implementation [34].

The population to be evaluated is sent via the RMI interface to the ‘Evaluation Distributor’ server running on the cluster’s head node. This then spawns a separate instance of the relevant Evaluator module for each member of the population, each scheduled to run on a different cluster node. Once each individual has been evaluated, the Evaluation Distributor sends the fitness values back to the Evolutionary Engine.

### 3.3 Remote laboratory connection

As it is hoped that this research will eventually be concerned with the artificial evolution of complex features in real, chemical systems, the ability to couple the EE to a chemical reactor array is of paramount importance. Moreover, for ease of use, flexibility and modularity, that this reactor array can reside remotely to the evolutionary software is highly desirable.

Together with collaborators in the chemistry departments at the University of Leeds and the University of Manchester, a protocol for linking the Evolutionary Engine to a chemical reactor array was developed. In this way, the software can be initiated over the web interface and, depending on the problem script chosen, candidate populations can be evaluated through a chemical reactor array in a remote

<b>Number of genes per chromosome</b>	unlimited <sup>†</sup>
<b>Gene data types</b>	int, double, boolean (range and accuracy unlimited <sup>†</sup> )
<b>Number of objectives</b>	unlimited <sup>†</sup> (1 uses standard GA; > 1 uses NSGA-II)
<b>Objective data types</b>	int, double (range and accuracy unlimited <sup>†</sup> )
<b>Optimisation direction</b> (per objective)	max or min
<b>Target file</b>	uploadable if applicable
<b>Evaluation module</b>	user-defined
<b>Selection mechanism</b>	rank, tournament, roulette wheel
<b>Recombination rate</b>	[0,1]
<b>Recombination operator*</b>	n-point crossover, uniform crossover, uniform arithmetic crossover
<b>Mutation rate*</b>	[0,1]
<b>Mutation operator*</b>	flip, reset, gaussian
<i>* these parameters can be specified differently for different subsets of genes in the chromosome</i>	
<b>Population size (<math>\lambda</math>)</b>	unlimited <sup>†</sup>
<b>Number of children (<math>\mu</math>)</b>	unlimited <sup>†</sup>
<b>Replacement strategy</b>	$\mu, \lambda$ or $\mu + \lambda$
<b>Initial population</b>	uploadable if applicable
<b>Initialisation seed</b>	user-specifiable (0 for (pseudo)random)
<b>Distribution</b>	on, off

<sup>†</sup> but subject to language and hardware constraints

TABLE 3.1: Customisable options and user-specified operators for the Evolutionary Engine

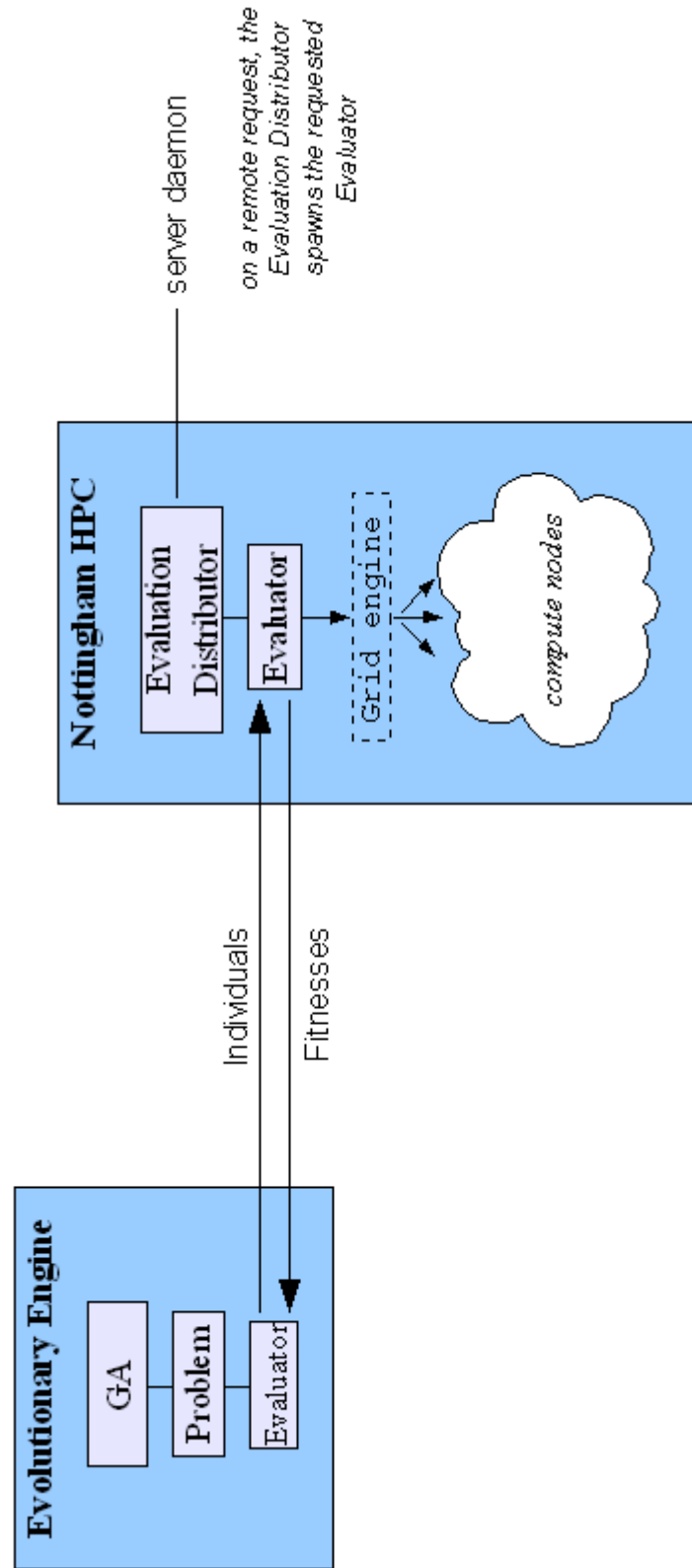


FIGURE 3.7: Graphical representation of the parallelisation capabilities of the Evolutionary Engine.

laboratory. The chemical reactor array itself is shown in figure 3.8. Essentially, it enables twenty different proportions of up to four reagents to be mixed concurrently and automatically – the syringe pumps that control the amount of each reagent, the parameters to the reaction chamber and the analysis of the output is all computer-controlled. The communication protocol between the Evolutionary Engine and the reactor array is a relatively simple one and is illustrated by figure 3.9.

### *Initial investigations*

A simple experiment was devised to illustrate that the system described above functions as expected. The problem under consideration was simple: Given four food dye solutions – red, blue, yellow and colourless, what proportions are required to manufacture a particular target resultant colour? An experiment was designed to optimise these parameters (the four proportions) such as to find a resultant solution that has some pre-defined target frequency value of maximum light absorbance.

The total volume of the resultant solution must be  $1ml$ . This constraint makes the genotype slightly more complex than if it were simply defining the volume of each reagent. The genotype comprises three real values,  $g_0$ ,  $g_1$  and  $g_2$ , each in the range  $[0,1]$  with a step size of 0.01. The volumes of each reagent,  $r_i$  are therefore:

$$r_0 = g_0$$

$$r_1 = (1 - r_0)g_1$$

$$r_2 = (1 - (r_0 + r_1))g_2$$

$$r_3 = (1 - (r_0 + r_1 + r_2))$$

In this way,  $\sum r$  is always equal to 1.

The chemical reactor array mixes the four reagents in the defined proportions. The maximum light absorbance level of each individual is automatically measured and compared to the target value (in this case,  $625nm$ ). It is this error that is minimised by the GA.

Figure 3.10 shows the progress of the initial generations of an experimental run of the system. The size of the population is set, by necessity, to twenty (there are twenty well plates on the chemical reactor array). Recombining individuals use uni-



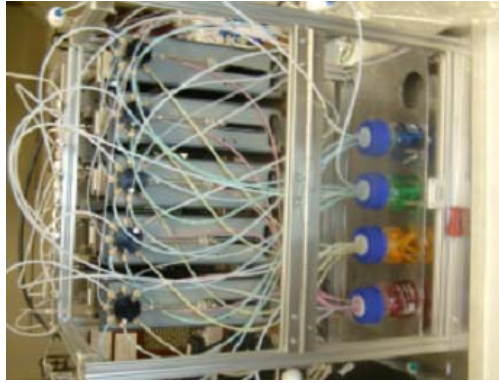
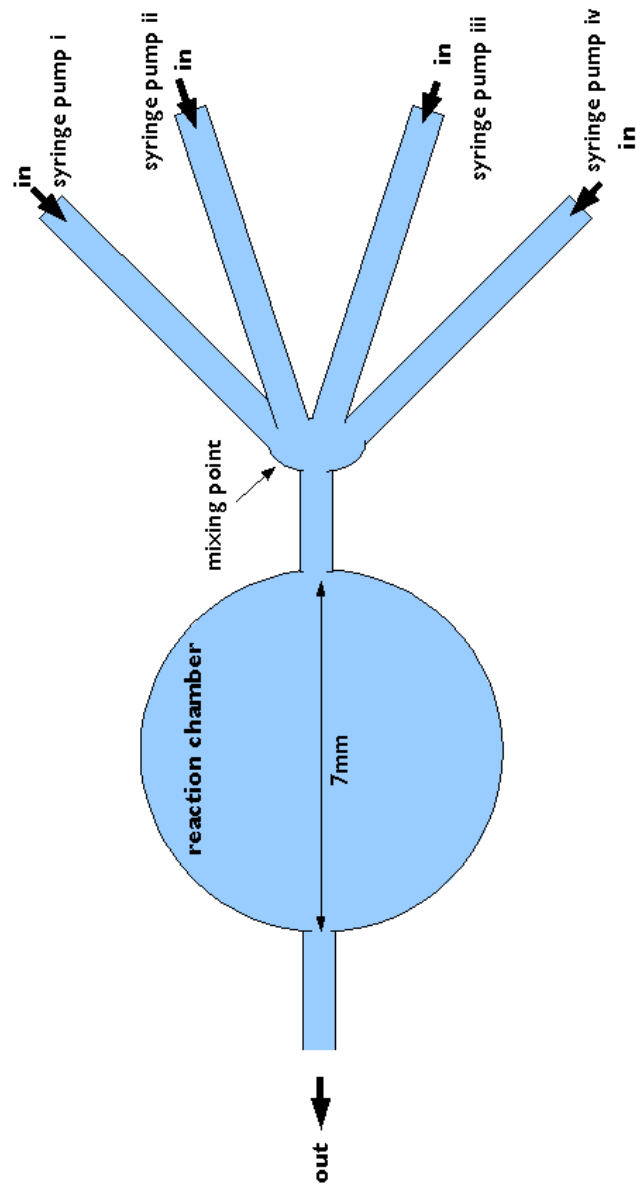


FIGURE 3.8: Diagram and photograph of the chemical reactor array to which the Evolutionary Engine is designed to connect

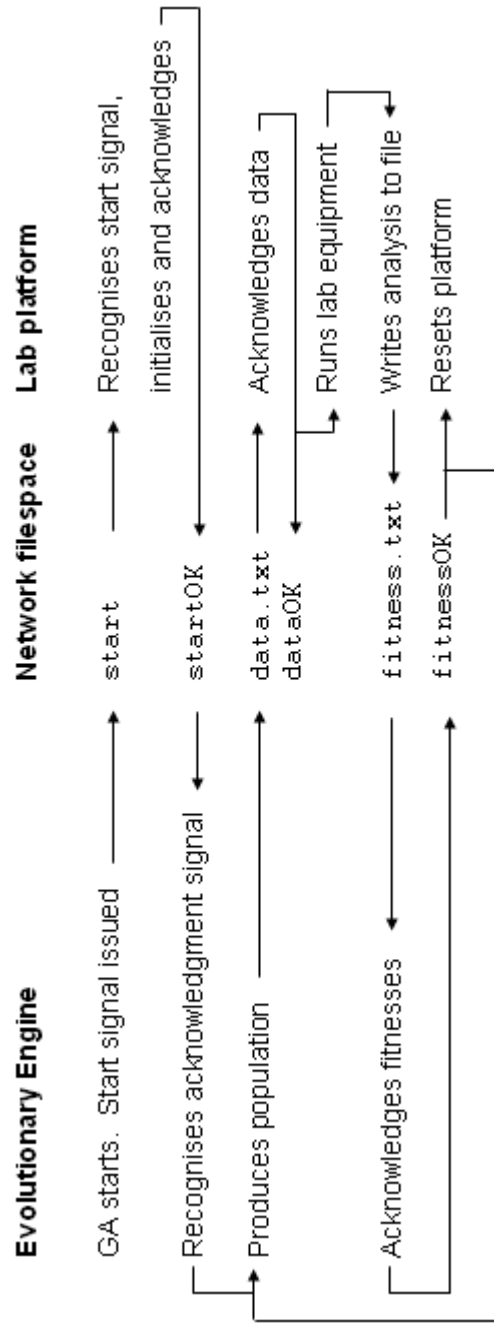


FIGURE 3.9: A schematic showing the communication protocol between the Evolutionary Engine and the remote chemical reactor array

form crossover with probability 0.9, and Gaussian mutation occurs with a probability of 0.1. The number of generations is dependent on the smooth running of the chemical reactor array – in these experiments, only a relatively small number (four) generations was possible, but this is still enough to demonstrate the success of the linking mechanism between the array and the EE, as well as demonstrating that the evolutionary computation methods are driving the system to its desired target behaviour. In this case, ‘fitness’ is defined as the numerical level of absorbance in the red region arithmetically divided by that in the blue region; in this manner, the search is directed to find the product with minimal absorbance in the blue region and maximal absorbance in the red (i.e., to the eye, the product will appear blue coloured). There is a clear increase of average fitness (that is to say, the average maximal light absorbance frequency is getting ever nearer the target value), and this can be clearly observed in the three graphs (figure 3.11) which show the absorbance spectrum of the entire population (twenty individuals) at the end of each generation of evolution. The x-axis in these figures represent the frequency of light (in the electro-magnetic spectrum), with the y-axis representing the level of absorbance of a given frequency of light by the solution under observation. It is evident that the population is converging onto a maximal absorbance in the region of the desired  $625nm$ . Though a simple test, the results above show that the evolutionary engine can be coupled, via a remote link, to a computer controlled chemical reactor array.

### 3.4 Conclusions

In this chapter, the specifics of the ‘Evolutionary Engine’ developed as part of this research was presented, particularly emphasising its scope for flexibility and ease-of-use. A brief, but powerful illustration was given of the EE’s capabilities to be connected to a physical laboratory reactor array system, giving rise to the potential for real-time, optimisation of complex chemical processes. However, it should be noted that the technical chemical hurdles needed to be overcome in order to realise this goal in anything more complex than the simple food dye test presented above are considerable, and have presented our chemist collaborators in Manchester with

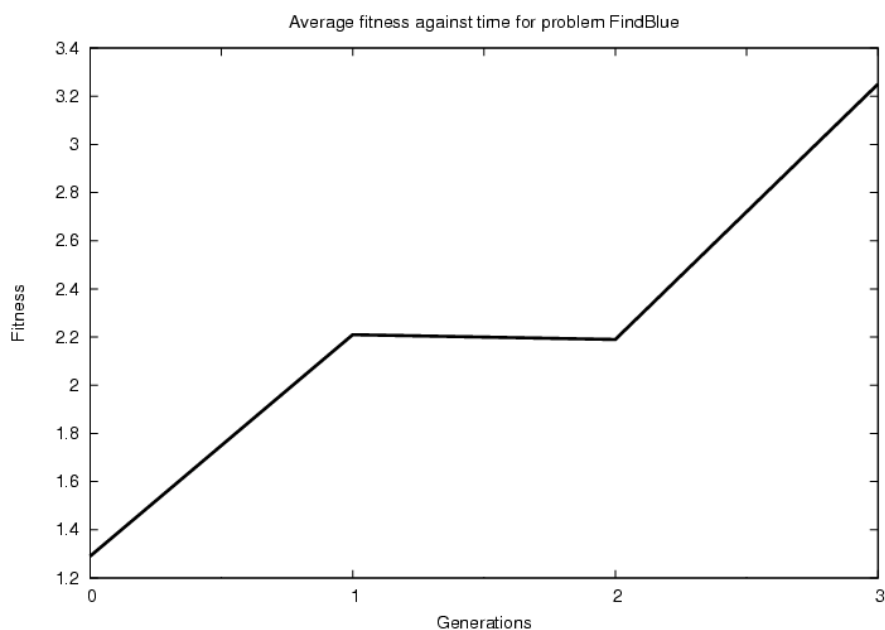


FIGURE 3.10: Evolution graph of the simple ‘find a colour’ experiment using the remote laboratory reactor array in conjunction with the Evolutionary Engine

significant difficulties. Thus, the rest of this thesis deals with digital systems and simulated physical complex systems in the belief that the software developed and insights gained into these systems will be transferrable onto ‘real’, *in vitro* systems, when the chemical setup is ready.

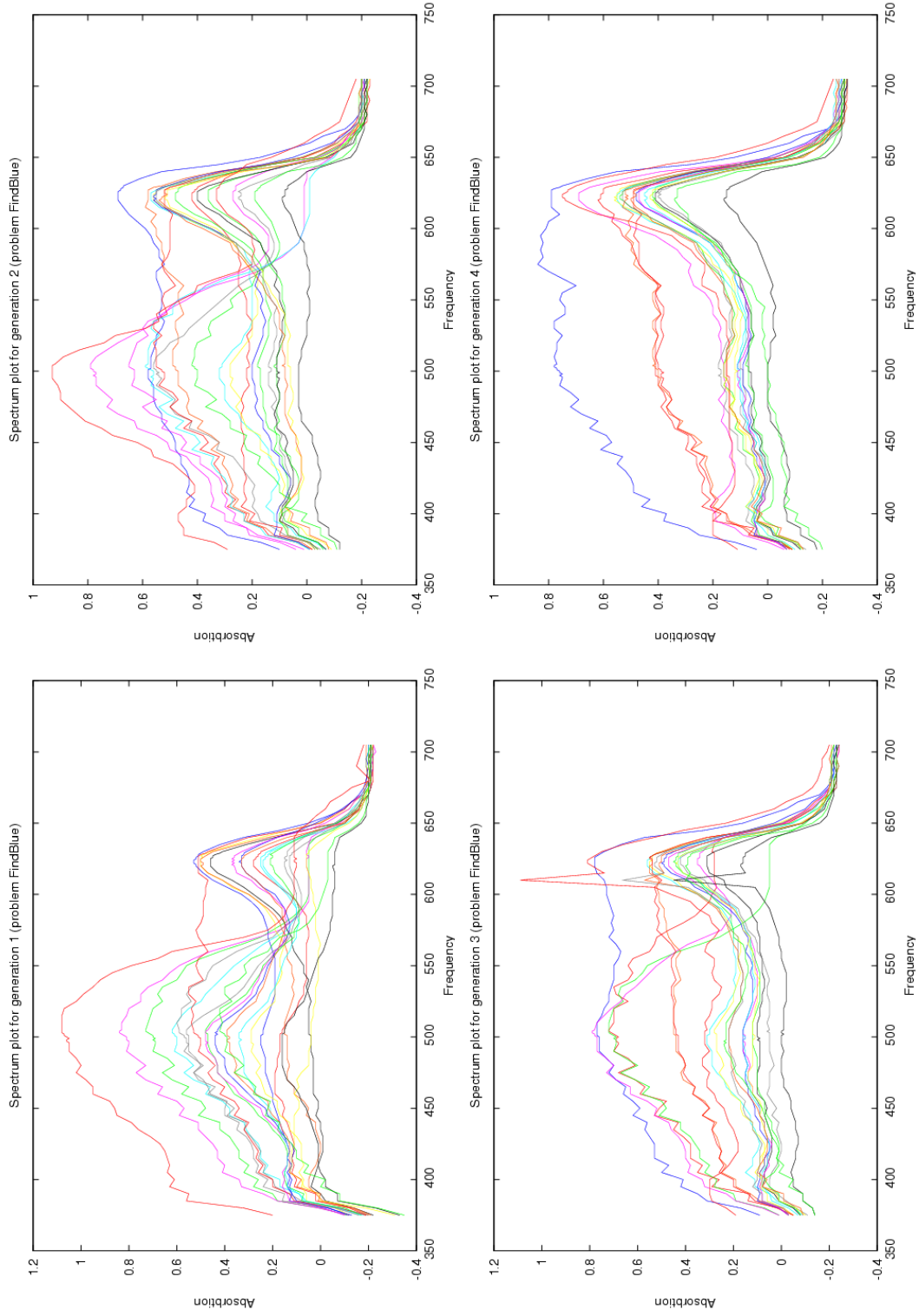


FIGURE 3.11: Absorbance spectra for generations 0-3 in the simple 'find a colour' experiment using the remote laboratory reactor array in conjunction with the Evolutionary Engine.

## CHAPTER 4

# A complex search space analysis protocol

In this chapter, the Evolutionary Engine is used to evolve a target behaviour in a particular complex system based on a cellular automaton. The importance of choosing a well-suited fitness function is demonstrated, and the Universal Similarity Metric is introduced and subsequently verified as a particularly successful measure of fitness in this problem. The results are repeated with two standard local search algorithms in an attempt to validate the hypothesis that a genetic algorithm is the more appropriate method for solving such problems.

Cellular automata (CA) have often been used to model natural processes such as gas diffusion systems [19], chemical reactions [66] and wave propagation in so-called ‘excitable media’ [130]. These systems usually have a number of numerical inputs that determine the nature of the spacio-temporal behaviour of the system, often captured by a graphical pattern. Genetic algorithms have been used to design CA-based systems [16, 86, 107, 108]. The work presented in this section (and published in [114]) deals specifically with using the Evolutionary Engine described in chapter 3 to evolve the parameters to a CA model such as to match a pre-specified target behaviour. This work serves as i) a proof-of-concept that the EE can effectively solve such parameter optimisation problems and more importantly, ii) to introduce the Universal Similar Metric as a potentially suitable, problem-independent fitness function.

## 4.1 Cellular automata - a description

Cellular automata are of particular interest, as they demonstrate the richness and power of behaviours and functions that can be produced using only very simple components driven by very simple rules. Homogeneity, massive parallelism, local cellular interactions and both synchronous and asynchronous models of rule execution are some of their most prominent features, which allow scientists to understand a variety of phenomena in, to name but a few, the physical, chemical, biological, social and information sciences.

A cellular automaton is essentially an infinite, regular grid of cells, each of which can be in one of a finite number of states – the set of states may be as simple as a binary white/black identifier, or may define a large number of ‘colours’ or states. The nature of the grid itself must also be defined; again, this may be as simple as a one-dimensional row of cells, but more usually it is a two-dimensional lattice which could be square, triangular or hexagonal in shape. The most complex style of CA systems may define  $n$ -dimensional grids. At a given time step,  $t$ , the state of a given cell is a function of that cell’s neighborhood at  $t-1$ . There are a number of possible definitions of a neighbourhood in CA systems. For example, the *Moore* neighbourhood uses the eight surrounding cells of the cell in question for the update process, using these eight states as input to the update function. A *von Neumann* neighbourhood only uses the four cells that are strictly adjacent to the central cell. A *Margolus* neighbourhood divides the grid into groups of four cells, to which the update function is applied completely locally (i.e., using only the information in this group of four cells); so as to allow propagation through the grid, the actual grouping of cells (in the 2x2 arrangement) changes on each update.

In [131], Wolfram presents extremely comprehensive analyses of various CA-based systems. He describes as “elementary cellular automata”, those systems which are based on a one-dimensional lattice, using a nearest neighbourhood – that is, the state of a given cell is a function of the cell immediately to its left and to its right. Wolfram enumerates all the possible rules (256) of such systems, and demonstrates the many and diverse behaviours of these simple systems. In figure 4.1, we show an example

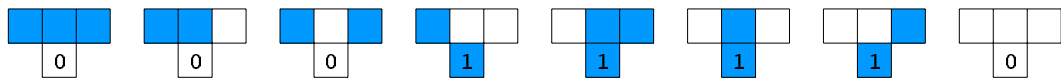
of the automaton produced by Wolfram’s “Rule 30”. This rule numbering system, described in [131], works as follows. Using a one-dimensional, nearest neighbour CA, the state of a given cell (1 or 0) is a function of the cell itself at the previous timestep, that of its left neighbour, and that of its right neighbour. This means there are  $2^3 = 8$  possible configurations for which a resultant state must be specified (on-on-on, on-on-off, on-off-on, etc., which can in turn be represented in binary as 111, 110, 101, etc.). Wolfram’s “Rule 30” is that CA for which the sequence of resultant states equates to decimal 30 when read in binary (00011110). That is to say, the first possible arrangement of ‘left neighbour on, self on, right neighbour on’ (or 111 or short) results in self being ‘off’ or ‘0’ at the next time step; the second arrangement (on-on-off or 110) results in ‘off’ or 0, etc., with all eight results spelling out 00011110 – the binary representation of the number 30. Some further examples: rule 0 is the rule for which all eight configurations result in ‘off’ (00000000); rule 255 is the rule for which all eight configurations result in ‘on’ (11111111); rule 31 will be very similar to the rule 30 described above, but the final configuration (off-off-off or 000) will result in on (1) rather than off (0). Figure 4.1 shows the “Rule 30” CA in operation, when seeded with an initial configuration of off-on-off. Each subsequent row describes the next time step of the system. Wolfram’s system is a simple, neat, and very effective method of evaluating and classifying all possible elementary cellular automata.

Perhaps the most famous CA system is the “Game of Life” discovered in 1970 by JH Conway [38]. Rather more complex than the simple CAs considered above, this model is based on an infinite two-dimension binary grid, with each cell either “live” or “dead”. The Moore neighbourhood is used (i.e. all eight neighbours are considered in the update function). At each time step, the following four rules are simultaneously applied:

1. Live cells with less than two live neighbours die
2. Live cells with more than three live neighbours die
3. Live cells with two or three live neighbours live
4. Dead cells with precisely three live neighbours become alive



Rule:



Example behaviour (seeded from a single '1' cell):

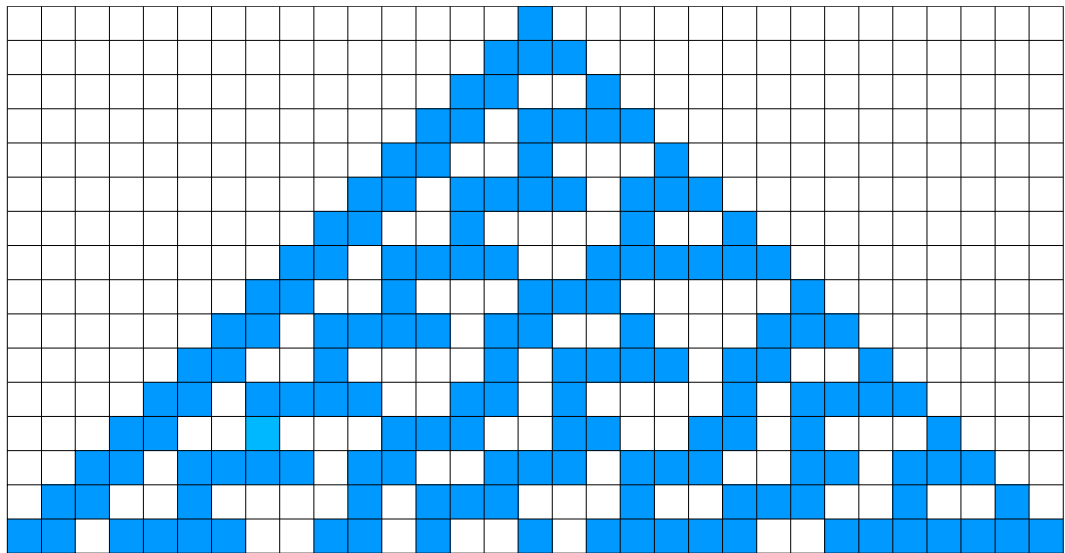


FIGURE 4.1: The elementary cellular automaton system produced using Wolfram's "Rule 30"

From this simple set of rules, an extraordinary wealth of functions can be produced, including reproductions, oscillations, gliders, pulsars and many others. In fact, the Game of Life is Turing-complete (i.e. it can perform any Turing computable algorithm). Cellular automata are clearly a very powerful class of computational device, and their ability to produce a large variety of complex behaviours from a collection of simple rules is particularly interesting to those researching the origins of life.

In the work presented below, we consider a particular CA-based system from the *NetLogo* [123] library. Named *Turbulence*, the model is used for investigating the relationship between turbulence, laminarity and viscosity of a fluid flowing through a pipe, and how the roughness of the pipe surface effects the fluid's behaviour. The pattern produced by the model captures the *entire* spacio-temporal behaviour of the system – that is to say, the lattice is one-dimensional, with each row of the image representing the next time step (as in the “Rule 30” system described above).

The model takes three, real-valued parameters, each of which can be seen to have a physical analogy in the system it is modelling. For each parameter described below, lower and upper bounds are quoted, as well as an indication of the discretisation step size of each parameter. The *initial-turbulence* [0,100,0.5] parameter specifies how disturbed the fluid is on its initial entry into the virtual pipe; parameter *coupling-strength* [0,1,0.1] is a measure of fluid-fluid interaction, essentially, a measure of the viscosity of the fluid; parameter *roughness* [0, 0.025, 0.001] is self-explanatory, governing the roughness of the inside of the pipe.

Each cell has a state value ranging continuously from 0 to 1.5, where 0 is the greatest degree of turbulence, and 1.5 is the greatest degree of laminarity. The transition rules are applied sequentially as follows:

1. Diffusion: The state of a given cell is averaged with its nearest neighbours on each side. The degree of influence that the cells have on one another is determined by the *coupling strength* variable (an approximation of the physical attribute of viscosity).
2. The state of each cell is modified according to the *roughness* parameter, i.e., the rougher the virtual pipe, the more turbulent the cell will become.

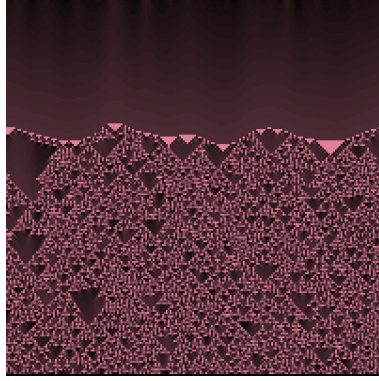


FIGURE 4.2: A example pattern from the *Turbulence* cellular automaton model. The pattern shows the entire spacio-temporal behaviour of the system, with each row representing a time step. This pattern was produced using parameter set  $\langle 20, 0.8, 0.005 \rangle$ .

An example behaviour pattern is shown in figure 4.2, where turbulence is represented by brightness, with darker areas representing more laminar regions.

## 4.2 Fitness

The aim of this work is to search for the parameters to the CA system such that a pre-determined target behaviour can be reproduced. In the system under consideration (the *Turbulence* model described above), behaviour is represented by a graphical pattern. In this case, therefore, we can draw an analogy between the system's behavioural and visual similarity. Hence, to evolve a target behaviour, we must have some method of measuring visual similarity between target and evolved behaviours in a numerical fashion.

The Universal Similarity Metric was first proposed in 2003 [72]. It is a compression-based mechanism for measuring the similarity between two general objects based on Kolmogorov complexity. Kolmogorov complexity is defined as the length of the shortest program for computing  $o$  by a Turing machine [73]. The definition can be extended to a conditional measure,  $K(o_1 \mid o_2)$ , which defines how much further information is needed to produce object  $o_1$ , given the information contained within  $o_2$ , otherwise known as the 'information distance'. Usually, a normalisation term is applied giving

the ‘Normalised Information Distance’:

$$NID(o_1, o_2) = \frac{\max\{K(o_1|o_2), K(o_2|o_1)\}}{\max\{K(o_1), K(o_2)\}} \quad (4.1)$$

Intuitively, it is clear that if no further information is needed, i.e., an information distance of zero, the two objects must be the same. Li et al. [72] showed that

$$K(o_1|o_2) + K(o_2) = K(o_1 \cdot o_2) \quad (4.2)$$

where  $\cdot$  represents concatenation. Using this substitution, we therefore have the alternative expression of Normalised Information Distance:

$$NID(o_1, o_2) = \frac{\max\{K(o_1 \cdot o_2) - K(o_2), K(o_2 \cdot o_1) - K(o_1)\}}{\max\{K(o_1), K(o_2)\}} \quad (4.3)$$

Kolmogorov complexity is not directly computable, hence a commonly used approximation is to use a compression algorithm to remove as much information redundancy as possible, and then to use the length of this compressed object as an approximation to Kolmogorov complexity. Hence, we can now calculate the ‘Normalised Compression Distance’, which is the measure used in the Universal Similarity Metric. Hence, if  $|C(o)|$  is the length of  $o$  when compressed by some compression algorithm,  $C$ :

$$USM(o_1, o_2) = \frac{\max\{|C(o_1 \cdot o_2)| - |C(o_1)|, |C(o_2 \cdot o_1)| - |C(o_2)|\}}{\max\{|C(o_1)|, |C(o_2)|\}} \quad (4.4)$$

Clearly, a lower value of the USM indicates a smaller information distance and thus a greater degree of similarity. When used in the context of an optimisation, therefore, it is a minimisation problem (i.e., lower = fitter).

Although the “No Free Lunch” theorem [132] states that there is no algorithm/solver that can produce solutions of statistically identical quality for all problem instances, there is still merit in research into algorithms that are generalisable in so far as they can produce *reasonable* results for *most* problem instances. This has the great advantage that the problem solver in question can be adaptable to a wide range of inputs within this problem domain of object similarity. For this reason, having a ‘universal’ fitness function may prove to be advantageous.

The USM has been particularly widely used in protein structure comparison problems [5, 69] where atomic contact maps are compared for similarity, but a number of

diverse uses of the metric are also evident in the literature, including [74] where it is used to compute a similarity matrix of a number of musical melodies. We investigate below whether the USM may be an appropriate method for comparing target and evolved behaviours of the CA system described above.

### 4.3 Robustness verification

Finding a reliable method of predicting when a GA will be an effective method of optimisation is still an open topic of research in evolutionary computation theory. The nature of complex systems means that the mapping between the genotype that specifies a behaviour and the actual realisation of that behaviour – the phenotype – may not be one-to-one, and may be highly non-linear, counter-intuitive and even stochastic. There is then a further relationship between this phenotype and the numerical fitness attached to it by the objective function. If there is not a clear correlation here, i.e., if the objective function cannot effectively differentiate between dissimilar phenotypes and cannot well-classify similar phenotypes, then the selection pressure introduced into the search will be meaningless or even misleading. Moreover, for the mutation to be of any use in ‘fine tuning’ the solutions found by crossover, there must be a clear correlation between genotype and fitness – if behavioural specifications that are only a few mutation steps away result in wildly different outward behaviours, effective search will be very difficult indeed.

Due to the complex nature of the genotype–phenotype–fitness mapping in the problems with which we are dealing, a method of verifying the efficacy and robustness of a fitness function is of great importance. In [114], we propose a two stage protocol for a process to verify whether a given fitness function could accurately direct a search in such complex dynamics. This two stage process involves both cluster analysis and fitness-distance correlation, which verifies the complex relationships described above (and as shown in figure 4.3).

Fitness Distance Correlation [63] is a measure of how effectively the fitness of an individual correlates to its genotypic distance to a known optimum. In other words, given two different genotypes, FDC measures the correlation of the (numerical)

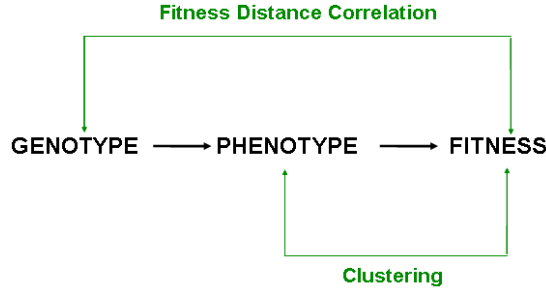


FIGURE 4.3: Mappings and analysis methods

distance between these genotypes against the value assigned by the objective function. If there is only a weak relationship between these two values, a parameter optimisation GA, or for that matter any metaheuristic based on the same representation, will have very little effect. Hence, FDC analyses the genotype–fitness relationship.

The authors of [125] suggest that problems can be classified into three distinct classes of difficulty – misleading (where fitness actually increases with genotypic distance from the known optimum), difficult (where little correlation is observable) and easy (where fitness clearly increases as the genotypic distance decreases). Although some sensible criticisms have been made of FDC (the authors of [2] construct an example problem where FDC’s failure to take into account the genetic operators themselves means it is not an effective measure of problem difficulty), its continued use within the literature [63, 125, 128] shows it remains a largely successful methodology.

We must also analyse the phenotype–fitness relationship, in other words, we must verify that the objective function can effectively differentiate between dissimilar phenotypes and well-classify similar phenotypes for the purpose of effective selection. If the fitness function cannot achieve this, a parameter optimisation GA will have difficulty evolving towards better solutions as the selection process will not have sufficiently accurate information to bias the search. For verification of the phenotype–fitness relationship, we use a hierarchical clustering method [47].

Such is the complexity of the genotype–phenotype mapping, that FDC cannot be guaranteed to give a completely accurate picture. Indeed, in our application, the objective function itself is also only an approximation of two individuals’ phenotypic

similarity. For these reasons, relying on only one of FDC or clustering to validate an objective function would not be adequate. Hence, we use both methods to show that a given fitness function is suitable for use in a particular problem.

#### 4.3.1 Datasets

It is important that the dataset used in this protocol is representative of the whole search space, whilst still being of a manageable size. Hence, the search space is systematically sampled by binning each of the  $n$  parameters of the genotype into  $b$  bins across their entire range. Every combination of these  $b^n$  parameter sets are then run through the complex system to produce their corresponding phenotypes.

Considering the three genes in the *Turbulence* system, *initial-turbulence*  $[0,100,0.5]$ , *coupling-strength*  $[0,1,0.1]$  and *roughness*  $[0, 0.025, 0.001]$ , the number of bins,  $b$ , is set to six, giving a dataset size of  $6^3 = 216$  points. The entire dataset is listed in the appendix (A.1).

#### 4.3.2 Fitness Distance Correlation

FDC [63] is a statistical-based methodology which performs a correlation analysis given a known target solution and samples from the search space. Faced with a maximisation problem, a high positive correlation value is interpreted as indicating that the problem may be effectively optimised by a GA, whereas a low negative value suggests that GA optimisation might not be as effective. Correlation coefficients around zero are inconclusive, and a closer examination is usually necessary. The correlations are often represented by a scatter plot of fitness versus distance.

$$r = \frac{(1/n) \sum_{i=1}^n (f_i - \bar{f})(d_i - \bar{d})}{S_F S_D} \quad (4.5)$$

The formula for the derivation of a correlation value is shown in equation 4.5, where  $r$  is the correlation coefficient,  $n$  is the number of individuals under consideration,  $f_i$  is the fitness of individual  $i$ , and  $d_i$  is its distance to the nearest global optimum,  $\bar{f}$  and  $S_F$  are the mean and standard deviation of the set of fitnesses, and  $\bar{d}$  and  $S_D$  are the mean and standard deviation of the set of distances.

The definition of ‘distance’ is somewhat more involved when, as in this problem domain, there are a number of genes with significantly different ranges, even different data types. A simple Euclidean distance would clearly introduce a false weighting in favour of the genes that naturally take larger values (for example, a distance of 1 in the context of one gene may represent 1% of its total range, whilst in another gene, it may represent 50% of its total range). In order to provide a more meaningful distance value, therefore, each genotypic distance is normalised into the range [0,1] and summed together to form a single value, that we term the *Normalised Combined Distance*,  $\hat{d}$  (equation 4.6 where  $G$  is the number of genes and  $g_j^i$  represents the  $j^{th}$  gene of individual  $i$ ). The terms  $g^{min}$  and  $g^{max}$  refer to the minimum and maximum values allowed by the range of the gene  $g$ .

$$\hat{d}_i = \sum_{j=1}^G \left( \frac{g_j^i - g_j^{target}}{g_j^{max} - g_j^{min}} \right) \quad (4.6)$$

An arbitrary member of the dataset is chosen as a candidate target for the FDC analysis. The corresponding phenotype is shown in figure 4.2, and was produced using parameters {20, 0.8, 0.005}.

A Fitness Distance Correlation plot ( $F$  against  $\hat{d}$ ) of the 216-piece dataset compared against the target described above is shown at figure 4.4. At first glance, there is no clear correlation, and the correlation coefficient itself – 0.165 – is indeterminate. However, a more detailed analysis can reveal a better indication of correlation:

Figures 4.5 - 4.7 show FDC plots for each parameter taken separately. Looking at the *initial-turbulence* parameter (figure 4.5), some correlations can be observed – there is a particularly strong correlation on the side of negative distance, and a correlation (considerably shallower, though steeper nearer the origin) can also be observed on the positive side of the x axis. Similarly, analysing the *coupling-strength* parameter (figure 4.6), a strong correlation on the side of negative distance can be clearly seen, as well as a high positive trend towards the extremity of the x axis (distance). Looking at the *roughness* parameter (figure 4.7), another strong correlation on the side of negative distance is obvious, whilst a shallower (though stronger nearer the origin) trend is also evident on the positive distance side. So, although the overall correlation is not



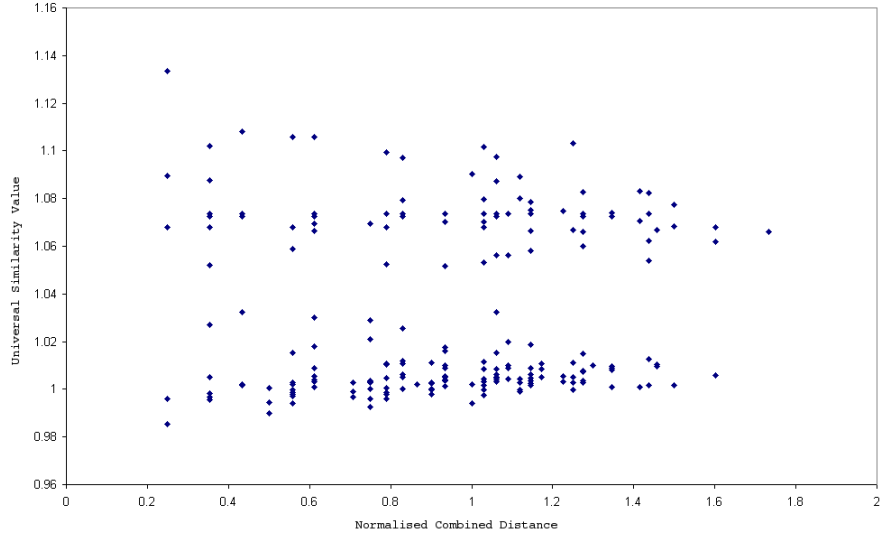


FIGURE 4.4: FDC scatter plot for the *Turbulence* system.  $r = 0.165$

terribly promising from the initial plot, a more detailed analysis shows that there are a number of well-correlated trends and so the USM may be an appropriate metric to use with this system and its associated genotypic representation.

### 4.3.3 Clustering

In order to further assess the proficiency of the USM as a fitness function and specifically to verify that it can effectively differentiate between dissimilar phenotypes and effectively classify similar phenotypes for the purposes of selection, a hierarchical clustering algorithm [47] is run over the dataset. Hierarchical clustering is a particularly popular implementation of the clustering process and has been used with very good results in similar classification work, such as that related to protein structure comparison [5].

Using the dataset described above, every pair of phenotypes (patterns) are compared using the USM-based fitness function, and a square distance matrix formed by the resulting values is obtained. From this matrix, an hierarchical cluster can be formed using a standard algorithm [14], as illustrated below.

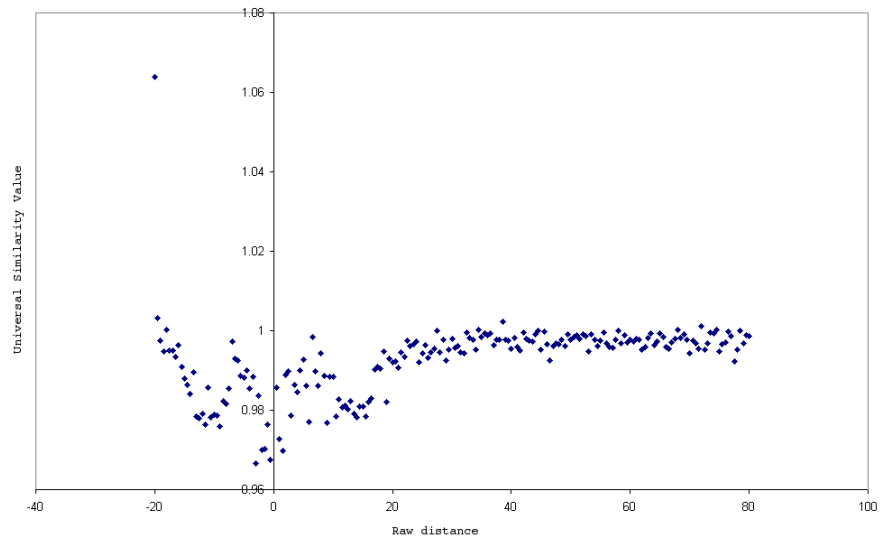


FIGURE 4.5: FDC scatter plot for the *Turbulence* system (*initial-turbulence* only).  $r = 0.143$

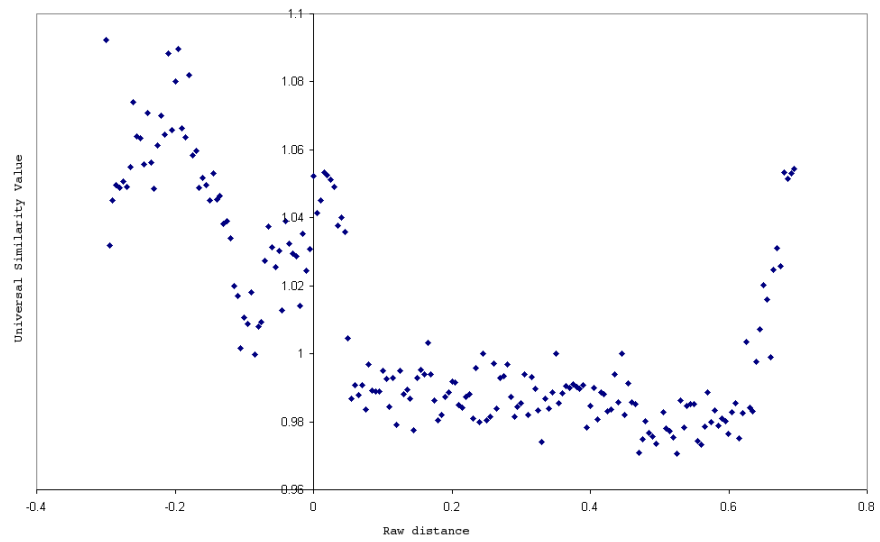


FIGURE 4.6: FDC scatter plot for the *Turbulence* system (*coupling-strength* only).  $r = -0.375$

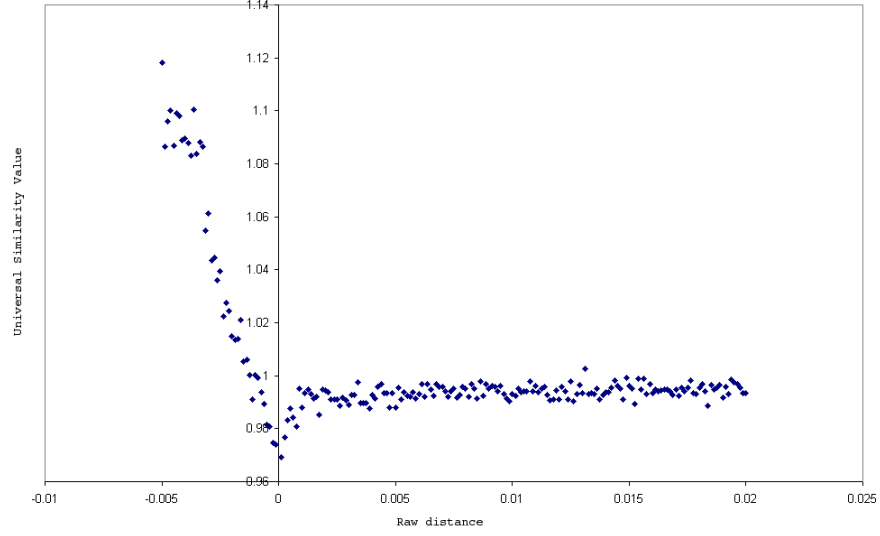


FIGURE 4.7: FDC scatter plot for the *Turbulence* system (*roughness* only).  $r = -0.151$

```

F = set of phenotypes;
for each pair of F as (Oi, Oj) do
    | M[i][j] = distance (Oi, Oj);
    | M[j][i] = distance (Oj, Oi);
end
initialise each sample as a separate cluster;
while unjoined clusters remain do
    | merge the two closest clusters;
    | calculate distance between the new cluster and every other cluster;
end
return clusters;

```

**Algorithm 2:** Pseudocode to show a standard hierarchical clustering algorithm

In this case, the *Complete Linkage* measure is used to calculate the inter-cluster distance, that is, the distance between clusters is calculated as the maximum distance between all pairs of samples. Inherently, this method is the least tolerant of noise (i.e., it gives particular weight to outlying points), and so if a good cluster can nonetheless be produced using this method, we can be particularly sure of the robustness of the USM's ability to differentiate between phenotypes. The output is

most conveniently visualised as a logarithmic tree [47]. By analysing each leaf of the tree (which correspond to a single point in the dataset), it can be seen that the USM effectively groups visually similar (which is analagous to the similarity of the CA's behaviour) together. Figure 4.8 shows the logarithmic tree representation of the 216-piece dataset described above annotated by example phenotypes representative of the clusters from which they come.

#### 4.4 Conclusions

In this chapter, a two-stage process was presented to serve as an helpful indication as to when a particular fitness function (in conjunction with a particular representation) may be successful in the context of a search algorithm. The results shown and analysed above – those for both Fitness Distance Correlation and Clustering – suggest that the Universal Similarity Metric should be an effective method for evolving target behaviour in graphical CA-based complex systems. Supported by this verification, in the next chapter, results are presented using this method on four target *Turbulence* patterns.

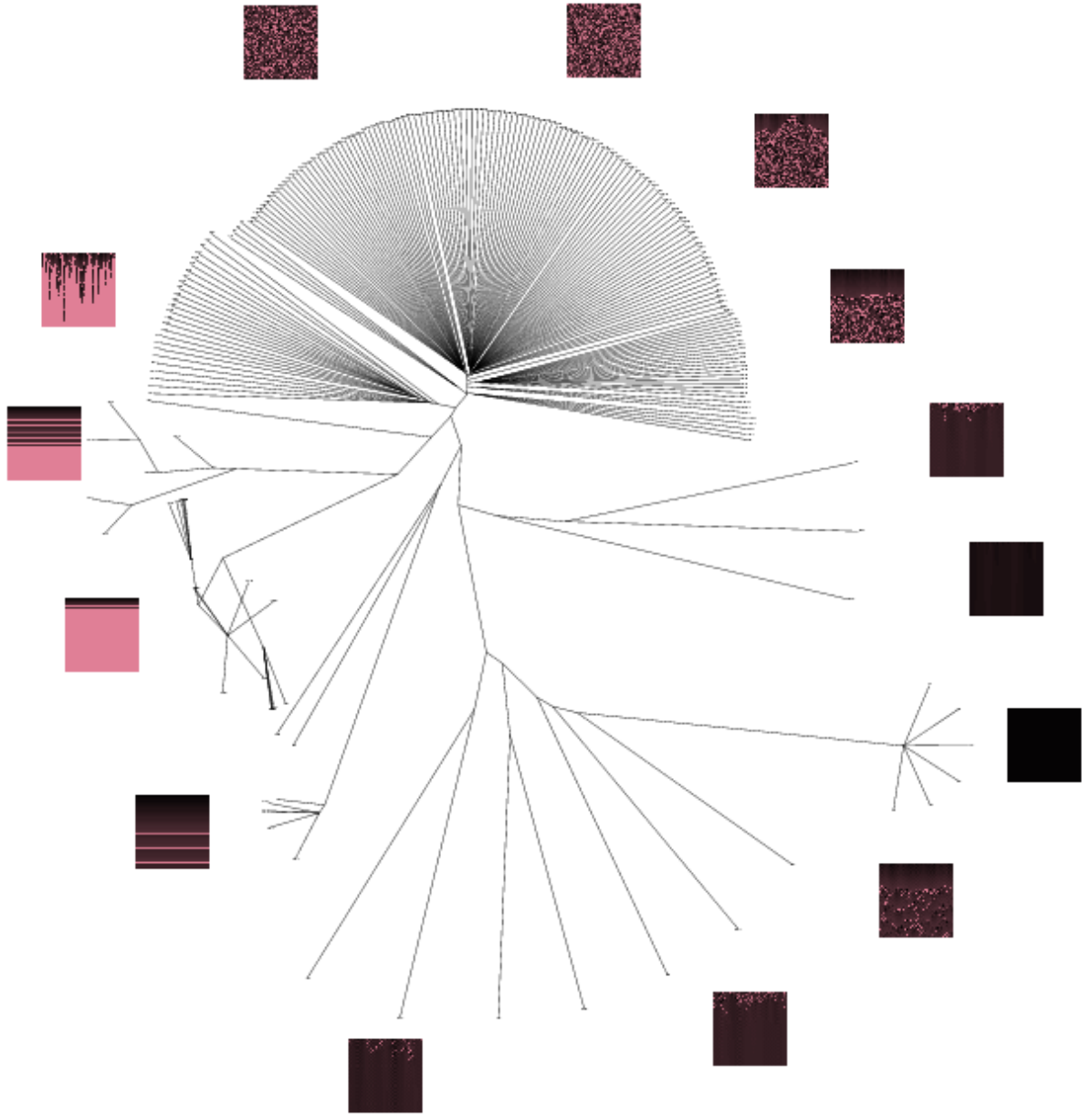


FIGURE 4.8: Logarithmic clustering tree showing the *Turbulence* dataset grouped according to phenotypic similarity (as calculated by the Universal Similarity Metric)

## CHAPTER 5

# Evolving cellular automaton systems

Having been given a reasonable indication by the robustness verification protocol presented in the previous chapter, that the USM is an appropriate objective function in this problem domain, it is used within a genetic algorithm whose aim is to generate a spatio-temporal behaviour ‘closest’ to some specified target image.

The aim of this work is to search for the parameters to the CA system such that a pre-determined target behaviour can be reproduced. In the system under consideration (the *Turbulence* model described above), behaviour is represented by a graphical pattern. In this case, therefore, we can draw an analogy between the system’s behavioural and visual similarity.

### 5.1 Experimental setup

The parameters described in the previous chapter are represented literally in the genotype, i.e., as floating point values. The roulette wheel parent selection method is used along with uniform crossover (probability 0.9) and Gaussian mutation (probability 0.1). These parameters are common ‘first attempt’ choices that can be tuned at a later stage. The emphasis here is on verifying that the USM can direct an effective search. The population size is twenty to match that for the chemical reactor experiments presented in 3.3 – the running of complex systems is often expensive, whether it be in terms of computer processor time or laboratory apparatus time, and so achieving good results with relatively small populations is of particular interest.

The GA was run ten times on each of four targets from across the search space, represented by four contrasting and visually distinctive phenotypes. The algorithm stops when there is no change in the average fitness of the population for ten generations. Results of each run for each target behaviour are shown in table 5.1, where the symbol  $\mu$  represents the arithmetic mean and  $c_v$  the coefficient of variation (that is, the standard deviation divided by the mean). Figures 5.1 - 5.4 show, for each problem, the target pattern, an evolution graph of fitness against time, and the highest quality evolved pattern from the ten experiments.

## 5.2 Results

In every case, the evolved patterns are very similar to the target behaviours. It is particularly interesting to note that for target Turb-20-0.8-0.005, the distance from the top-most edge of the pattern to the point where activity begins (representative of time) is very well reproduced, also the size distribution of the regions of inactivity (the dark areas) in Turb-40-0.2-0.005 and Turb-60-0.4-0 are closely matched. Looking at table 5.1, it is evident that an exact match of the genotype is not required to produce a well-matched phenotype. That is to say, the genotype-phenotype mapping is not one-to-one – a number of genotypes can result in similar, if not identical phenotypes. The fitness landscape is therefore multi-modal – there may be a number of solutions in different areas of the search space, all of which give a high fitness (but not necessarily equally high). These are common traits of complex systems, and features that can make them particularly hard to search; the use of an ‘intelligent-sampling’ metaheuristic such as a genetic algorithm is therefore essential.

The mean number of generations taken to reach the stopping condition does not vary significantly across the four different targets, though within the ten runs for each target, there is a reasonably large variety, particularly in the case of Turb-20-0.8-0.005, as represented by the largest  $c_v$  value of the four – 0.324 – that represents a range of values from 101 - 265. In general, however, the majority of experiments run for a number of generations close to the mean - exceptions such as Turb-20-0.8-0.005/Run 10 and Turb-60-0.4-0/Run 5 are relatively rare. The best fitness values within each

Target	Run	Generations	Average	Best	Winning individual		
Turb-20-0.8-0.005	1	101	$0.96389 \pm 0.00094$	0.96167	4.3	0.8	0.007
Turb-20-0.8-0.005	2	162	$0.96714 \pm 0.00080$	0.96539	9.7	0.9	0.006
Turb-20-0.8-0.005	3	172	$0.96654 \pm 0.00091$	0.96423	1.3	0.5	0.008
Turb-20-0.8-0.005	4	133	$0.96327 \pm 0.00087$	0.96131	4.2	0.8	0.007
Turb-20-0.8-0.005	5	137	$0.96322 \pm 0.00071$	0.96142	4.3	0.8	0.007
Turb-20-0.8-0.005	6	108	$0.96533 \pm 0.00046$	0.96391	3.1	0.8	0.007
Turb-20-0.8-0.005	7	119	$0.96707 \pm 0.00051$	0.96592	0.4	0.8	0.008
Turb-20-0.8-0.005	8	182	$0.96407 \pm 0.00071$	0.96218	4.4	0.8	0.007
Turb-20-0.8-0.005	9	122	$0.96592 \pm 0.00129$	0.96179	6.5	0.8	0.007
Turb-20-0.8-0.005	10	265	$0.96529 \pm 0.00090$	0.96309	2.7	0.8	0.007
	$\mu$	150	0.96517	0.96309			
	$c_v$	0.324	0.00155	0.00176			
Turb-20-0-0.005	1	110	$1.01328 \pm 0.00304$	1.00192	89.5	0.0	0.007
Turb-20-0-0.005	2	115	$0.99347 \pm 0.00156$	0.99013	19.5	0.0	0.006
Turb-20-0-0.005	3	128	$1.01575 \pm 0.00144$	1.01085	53.8	0.0	0.014
Turb-20-0-0.005	4	119	$1.01313 \pm 0.00201$	1.00805	61.7	0.0	0.006
Turb-20-0-0.005	5	129	$1.01283 \pm 0.00180$	1.00820	88.9	0.0	0.005
Turb-20-0-0.005	6	157	$0.99973 \pm 0.00175$	0.99637	13.0	0.0	0.018
Turb-20-0-0.005	7	110	$1.01275 \pm 0.00331$	1.00034	47.0	0.0	0.006
Turb-20-0-0.005	8	132	$0.99273 \pm 0.00297$	0.98542	17.6	0.0	0.006
Turb-20-0-0.005	9	185	$0.99242 \pm 0.00315$	0.98568	23.3	0.0	0.005
Turb-20-0-0.005	10	131	$0.98884 \pm 0.00249$	0.98193	20.5	0.0	0.005
	$\mu$	132	1.00349	0.99689			
	$c_v$	0.177	0.01091	0.01063			
Turb-40-0.2-0.005	1	115	$0.97089 \pm 0.00057$	0.96945	48.9	0.9	0.005
Turb-40-0.2-0.005	2	168	$0.95906 \pm 0.00043$	0.95809	41.5	0.2	0.006
Turb-40-0.2-0.005	3	114	$0.95912 \pm 0.00057$	0.95797	41.3	0.2	0.007
Turb-40-0.2-0.005	4	111	$0.96066 \pm 0.00038$	0.95983	45.3	0.2	0.007
Turb-40-0.2-0.005	5	178	$0.95846 \pm 0.00084$	0.95554	40.9	0.2	0.006
Turb-40-0.2-0.005	6	102	$0.95951 \pm 0.00061$	0.95800	41.6	0.2	0.007
Turb-40-0.2-0.005	7	206	$0.95860 \pm 0.00055$	0.95711	40.7	0.2	0.006
Turb-40-0.2-0.005	8	143	$0.95923 \pm 0.00034$	0.95851	38.6	0.2	0.007
Turb-40-0.2-0.005	9	195	$0.96938 \pm 0.00063$	0.96833	48.2	0.9	0.008
Turb-40-0.2-0.005	10	135	$0.96102 \pm 0.00093$	0.95870	41.0	0.2	0.008
	$\mu$	147	0.96159	0.96015			
	$c_v$	0.257	0.00477	0.00494			
Turb-60-0.4-0	1	184	$0.95716 \pm 0.00057$	0.95578	98.0	0.4	0.000
Turb-60-0.4-0	2	148	$0.95701 \pm 0.00066$	0.95513	45.5	0.4	0.006
Turb-60-0.4-0	3	106	$0.95707 \pm 0.00067$	0.95526	54.5	0.4	0.000
Turb-60-0.4-0	4	140	$0.95647 \pm 0.00069$	0.95501	66.9	0.4	0.001
Turb-60-0.4-0	5	222	$0.95677 \pm 0.00065$	0.95454	78.3	0.4	0.002
Turb-60-0.4-0	6	119	$0.95790 \pm 0.00064$	0.95626	45.1	0.4	0.009
Turb-60-0.4-0	7	125	$0.95642 \pm 0.00063$	0.95504	80.7	0.4	0.000
Turb-60-0.4-0	8	173	$0.95644 \pm 0.00069$	0.95505	95.9	0.4	0.000
Turb-60-0.4-0	9	134	$0.95792 \pm 0.00068$	0.95651	76.2	0.4	0.007
Turb-60-0.4-0	10	103	$0.95690 \pm 0.00070$	0.95510	62.2	0.4	0.007
	$\mu$	145	0.95701	0.95537			
	$c_v$	0.259	0.00057	0.00065			

TABLE 5.1: Statistical analysis of Turbulence system evolutions, using GA/USM.



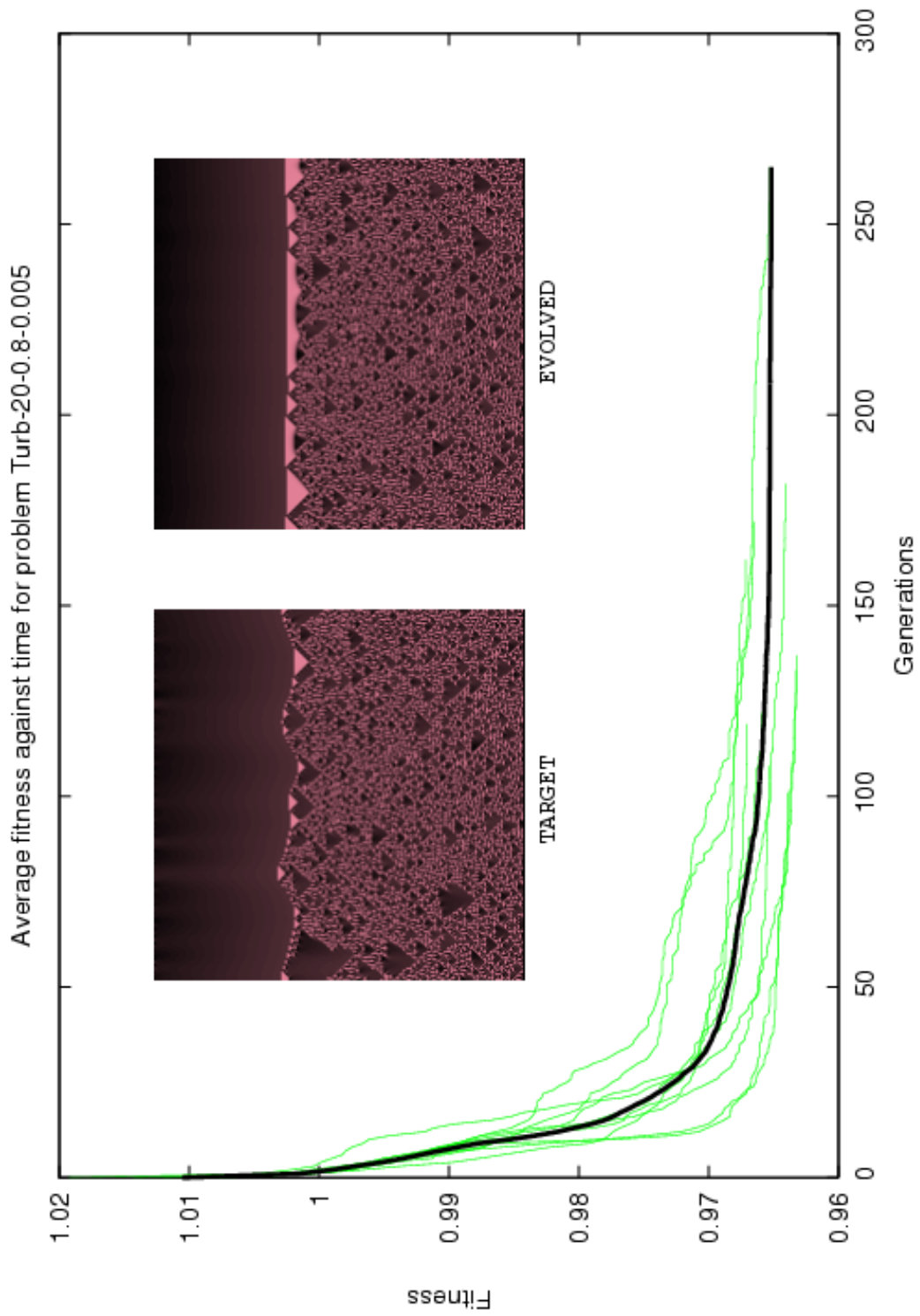


FIGURE 5.1: Graph of fitness against time, plus target and evolved behaviour for a GA functioning on problem Turb-20-0.8-0.005

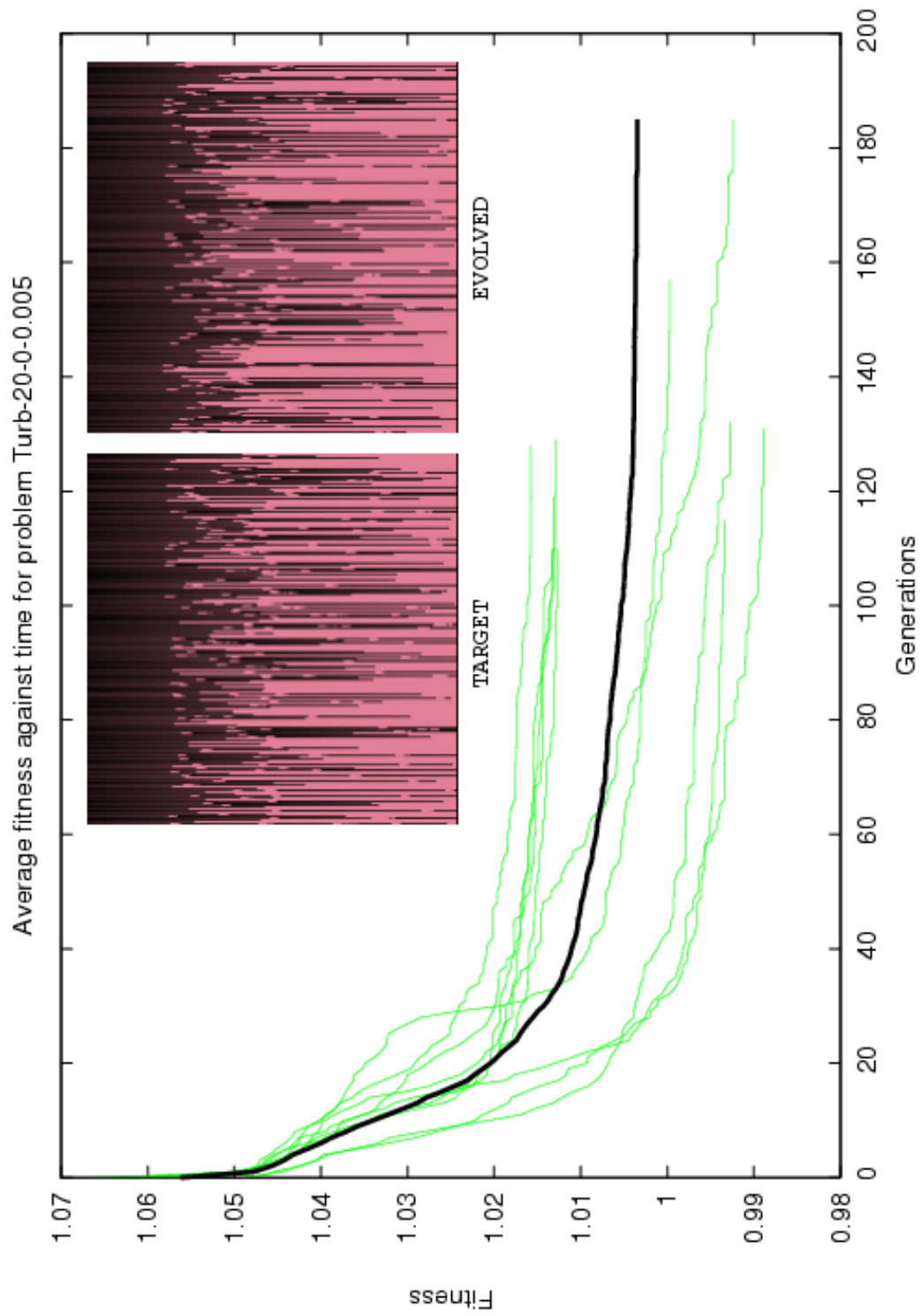


FIGURE 5.2: Graph of fitness against time, plus target and evolved behaviour for a GA functioning on Turb-20-0-0.005

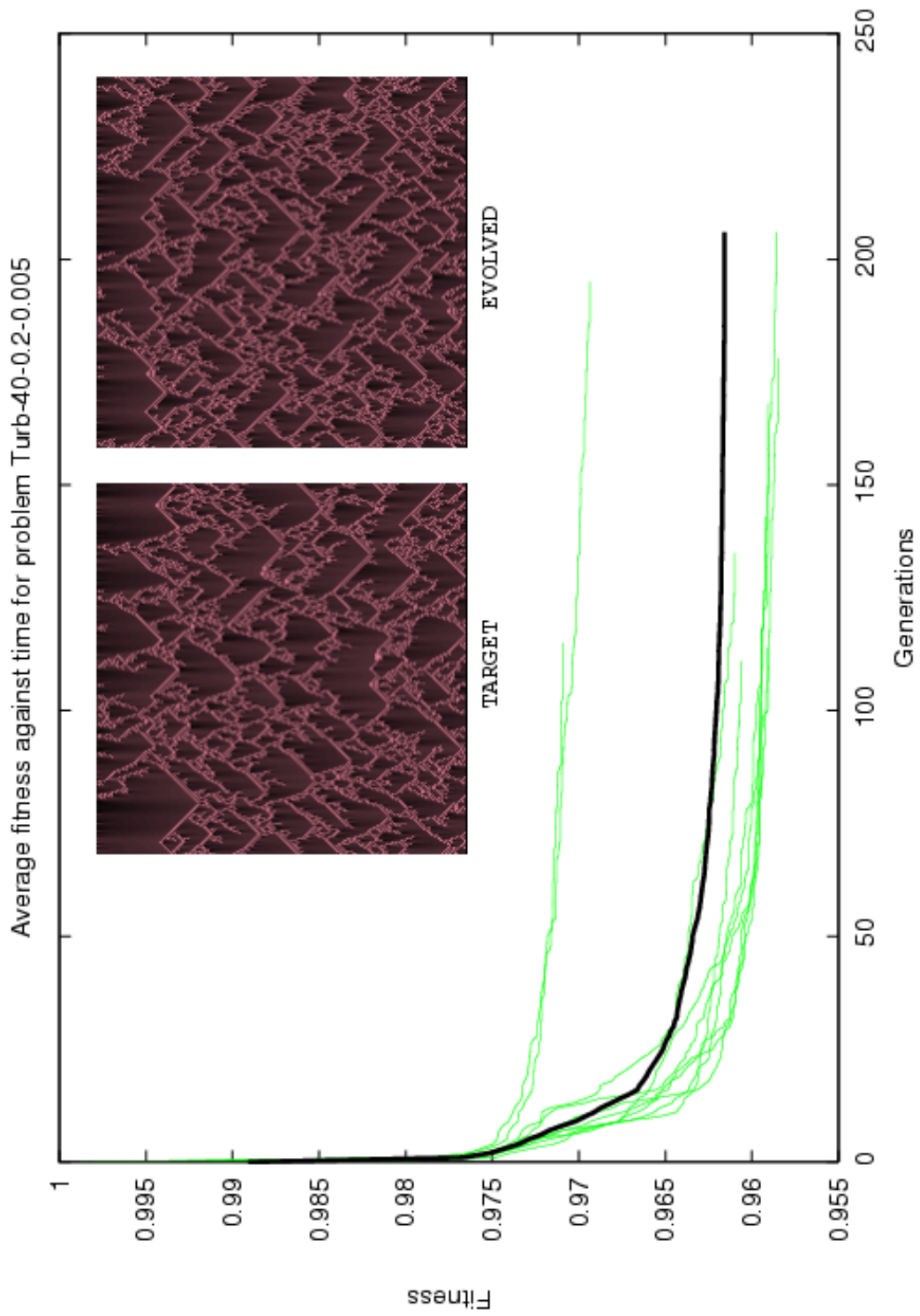


FIGURE 5.3: Graph of fitness against time, plus target and evolved behaviour for a GA functioning on problem Turb-40-0.2-0.005

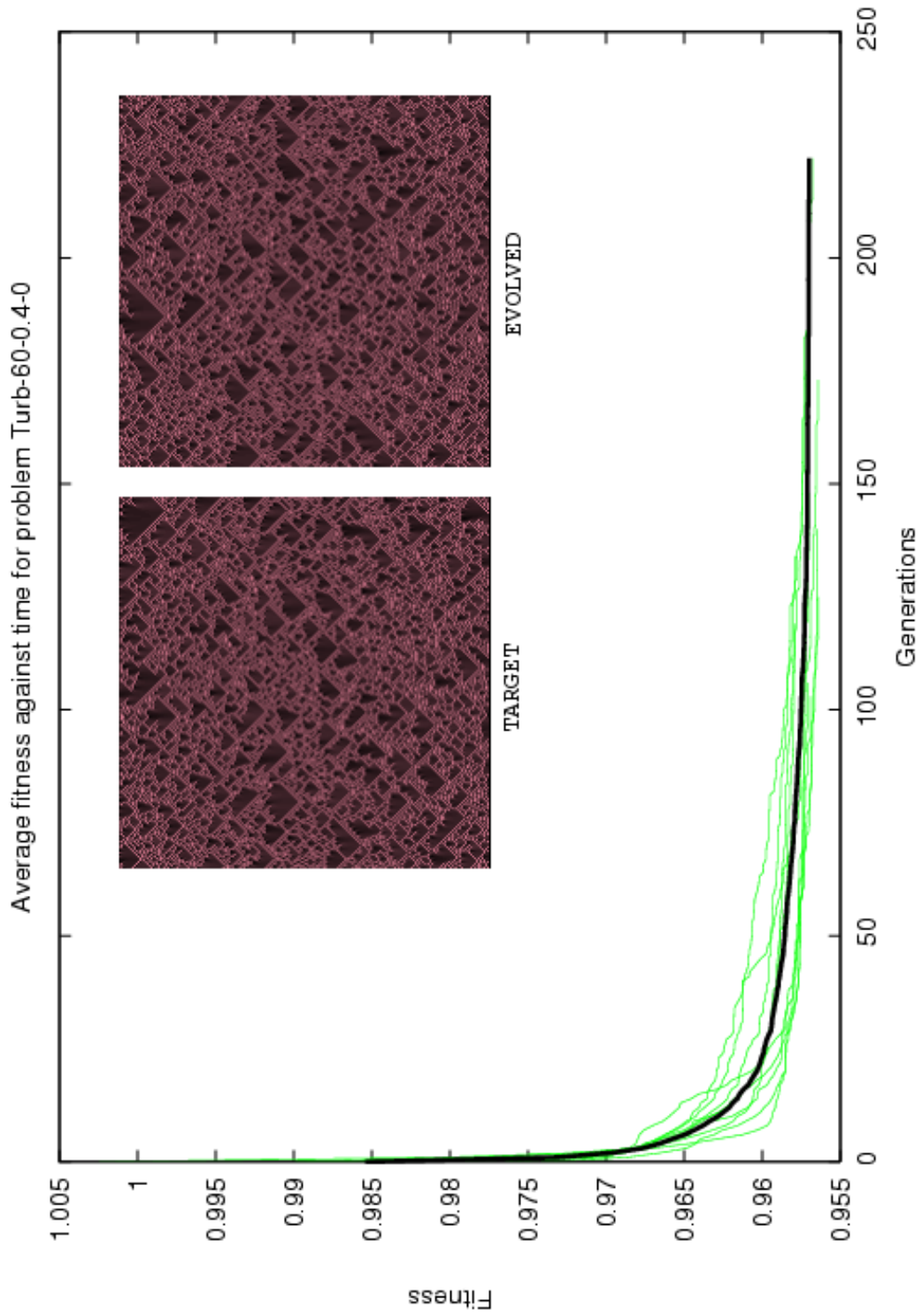


FIGURE 5.4: Graph of fitness against time, plus target and evolved behaviour for a GA functioning on problem Turb-60-0.4-0

experiment demonstrate very little variation, though Turb-20-0-0.005 is rather more variant (with a  $c_v$  of 0.01) than the other three (each with  $c_v$  values of a factor of ten or so smaller). This feature is also evident in the graphs (figures 5.1 - 5.4), where a greater spread of trajectories can be seen in the graph for Turb-20-0-0.005. It is important to note, however, that even given this spread, the resultant patterns are visually similar to their respective targets in every case, demonstrating the robustness of the methodology, as verified by the FDC/Clustering protocol described above.

### 5.3 Local search

Although, as mentioned above, it is long established that local search methods are not appropriate for the optimisation of particularly complex problems (hence our use of Genetic Algorithms in the first instance), and although the specific search algorithm itself is not the guiding concern of this research, it is important to ‘benchmark’ these results against a selection of simpler search mechanisms. First, a random search algorithm is implemented that simply samples individuals at random from across the search space, maintaining a memory of the ‘best’ found so far:

```

while stopping condition not fulfilled do
  | candidate = generate random individual;
  | if candidate > best then
  | | best = candidate;
  | end
end
return best;
```

**Algorithm 3:** Pseudocode to show standard random search procedure

Secondly, the popular ‘hill climbing’ algorithm is implemented that, from a randomly selected seed individual, examines an individual in its ‘neighbourhood’ (implemented through a single Gaussian mutation in one gene). If and only if this individual’s fitness exceeds the best found so far, it is adopted as the new winning individual:

```

candidate = generate random individual;
while stopping condition not fulfilled do
    | if candidate > best then
    |   | best = candidate;
    | end
    | candidate = mutate(candidate);
end
return best;

```

**Algorithm 4:** Pseudocode to show standard “hill climbing” search procedure

### 5.3.1 Random search

The average number of generations the GA took to reach a solution are listed in table 5.1 as  $\mu$ . The RS algorithm was run for this same number of iterations in each case. This may seem to be giving the GA an unfair advantage, as for each iteration the GA would sample  $\lambda$  points (where  $\lambda$  is the size of the population) where RS would only sample one. However, it is important to note that in the chemical optimisation problems to which we hope to connect this software at a later stage, the execution of one parameter set in the complex system (i.e., the chemical reactor array) is as costly as the execution of  $\lambda$  parameter sets, due to the reactor’s innate parallelism. To measure the success of these algorithms in terms of points sampled rather than iterations would be to ignore the fact that the GA, by its population-based nature, makes better use of the hardware. Hence, a ‘per iteration’ comparison is performed. Granted, alterations and more sophisticated versions of the local search methods could be employed, but so too could the parameters for the GA be tuned and refined; it is important to note again, that the focus of this thesis is on the evolution of complexity through appropriate similarity measures by *any* search mechanism.

Each experiment, as before, was run ten times. As before, figures showing the target, evolution graphs of fitness against time, and the best evolved individual are shown in figures 5.5 - 5.8. Details of each search are shown in table 5.2.

Looking at targets Turb-20-0.8-0.005 and Turb-20-0-0.005, it is clear that the Random Search results do not match the quality of those for the Genetic Algorithm,

Target	Run	Iterations	Fitness	Winning individual		
Turb-RS-20-0.8-0.005	1	150	0.97929	4.3	0.5	0.006
Turb-RS-20-0.8-0.005	2	150	0.98085	2.5	0.5	0.011
Turb-RS-20-0.8-0.005	3	150	0.97692	7.1	0.7	0.007
Turb-RS-20-0.8-0.005	4	150	0.97776	2	0.8	0.009
Turb-RS-20-0.8-0.005	5	150	0.97127	3.8	0.6	0.006
Turb-RS-20-0.8-0.005	6	150	0.97734	0.6	0.8	0.009
Turb-RS-20-0.8-0.005	7	150	0.97782	9.9	0.7	0.006
Turb-RS-20-0.8-0.005	8	150	0.97316	8.9	0.5	0.006
Turb-RS-20-0.8-0.005	9	150	0.97175	5.5	0.8	0.007
Turb-RS-20-0.8-0.005	10	150	0.9776	3.6	0.7	0.006
	$\mu$		0.97638			
	$c_v$		0.00322			
Turb-RS-20-0-0.005	1	132	1.03753	92.7	0.2	0.005
Turb-RS-20-0-0.005	2	132	1.02636	91.4	0	0.021
Turb-RS-20-0-0.005	3	132	1.02596	64.8	0	0.015
Turb-RS-20-0-0.005	4	132	1.03907	94.2	0	0.007
Turb-RS-20-0-0.005	5	132	1.00289	12.8	0	0.021
Turb-RS-20-0-0.005	6	132	1.02248	8.7	0	0.017
Turb-RS-20-0-0.005	7	132	1.02101	90.7	0	0.016
Turb-RS-20-0-0.005	8	132	1.03973	48.4	0.1	0.002
Turb-RS-20-0-0.005	9	132	1.01917	79	0	0.024
Turb-RS-20-0-0.005	10	132	1.03475	81.5	0	0.004
	$\mu$		1.02690			
	$c_v$		0.01115			
Turb-RS-40-0.2-0.005	1	147	0.96543	65.1	0.2	0.008
Turb-RS-40-0.2-0.005	2	147	0.96232	39.3	0.2	0.007
Turb-RS-40-0.2-0.005	3	147	0.96594	44.9	0.1	0.006
Turb-RS-40-0.2-0.005	4	147	0.96679	88.2	0.1	0.009
Turb-RS-40-0.2-0.005	5	147	0.96491	78.3	0.2	0.004
Turb-RS-40-0.2-0.005	6	147	0.96858	57.7	0.2	0.008
Turb-RS-40-0.2-0.005	7	147	0.96382	56.2	0.1	0.005
Turb-RS-40-0.2-0.005	8	147	0.96398	76	0.2	0.004
Turb-RS-40-0.2-0.005	9	147	0.96675	68.7	0.1	0.009
Turb-RS-40-0.2-0.005	10	147	0.96583	33.5	0.1	0.008
	$\mu$		0.96543			
	$c_v$		0.00185			
Turb-RS-60-0.4-0	1	145	0.96095	70	0.3	0.002
Turb-RS-60-0.4-0	2	145	0.95705	55.6	0.4	0
Turb-RS-60-0.4-0	3	145	0.96304	88.3	0.4	0.016
Turb-RS-60-0.4-0	4	145	0.96197	43.5	0.3	0.01
Turb-RS-60-0.4-0	5	145	0.96124	74.4	0.4	0.003
Turb-RS-60-0.4-0	6	145	0.96103	99.9	0.3	0.006
Turb-RS-60-0.4-0	7	145	0.95991	60.3	0.3	0
Turb-RS-60-0.4-0	8	145	0.9623	80.1	0.3	0.003
Turb-RS-60-0.4-0	9	145	0.96047	67.5	0.3	0
Turb-RS-60-0.4-0	10	145	0.96115	66.6	0.3	0.004
	$\mu$		0.96091			
	$c_v$		0.00169			

TABLE 5.2: Statistical analysis of Turbulence system random search

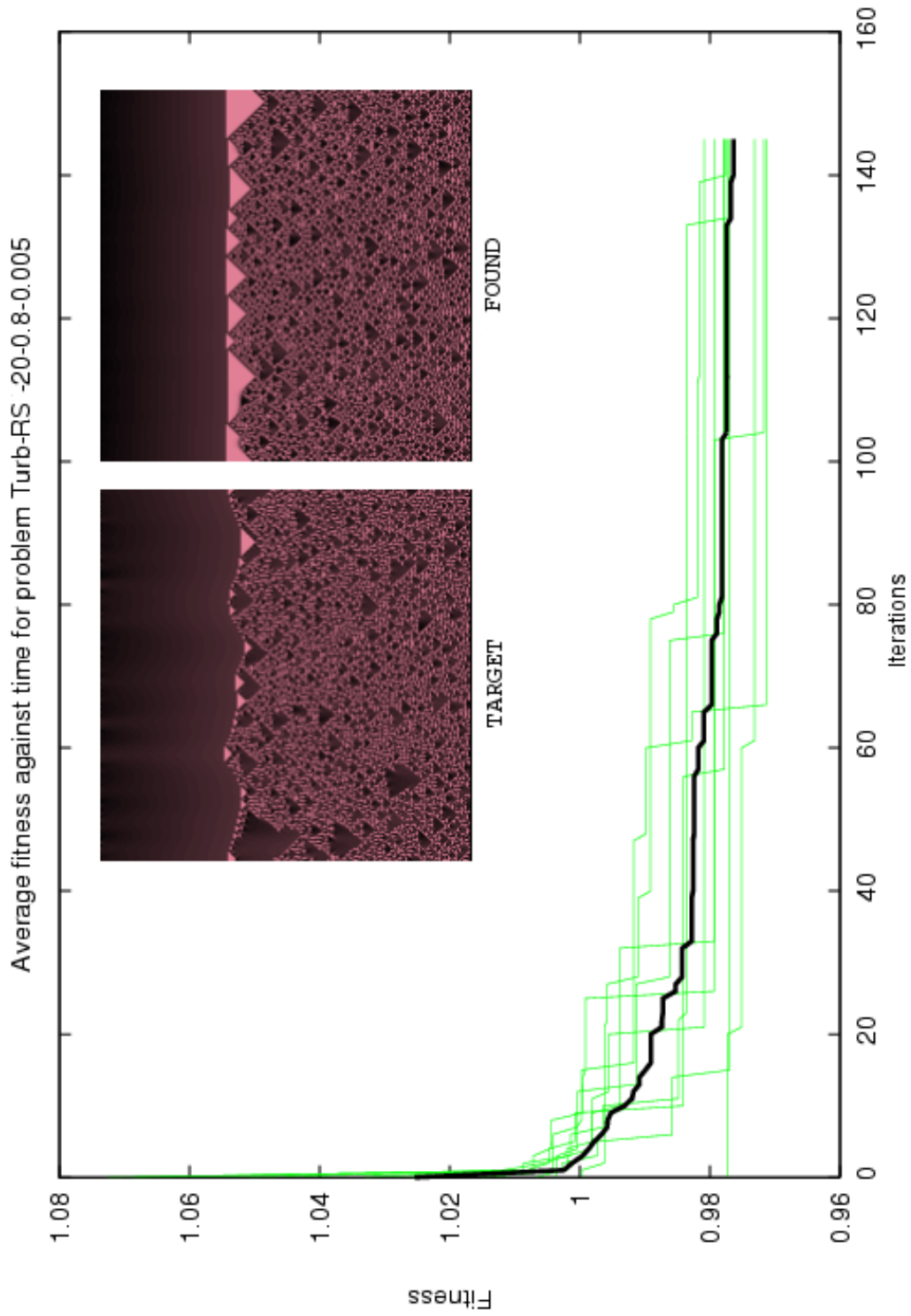


FIGURE 5.5: Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-20-0.8-0.005



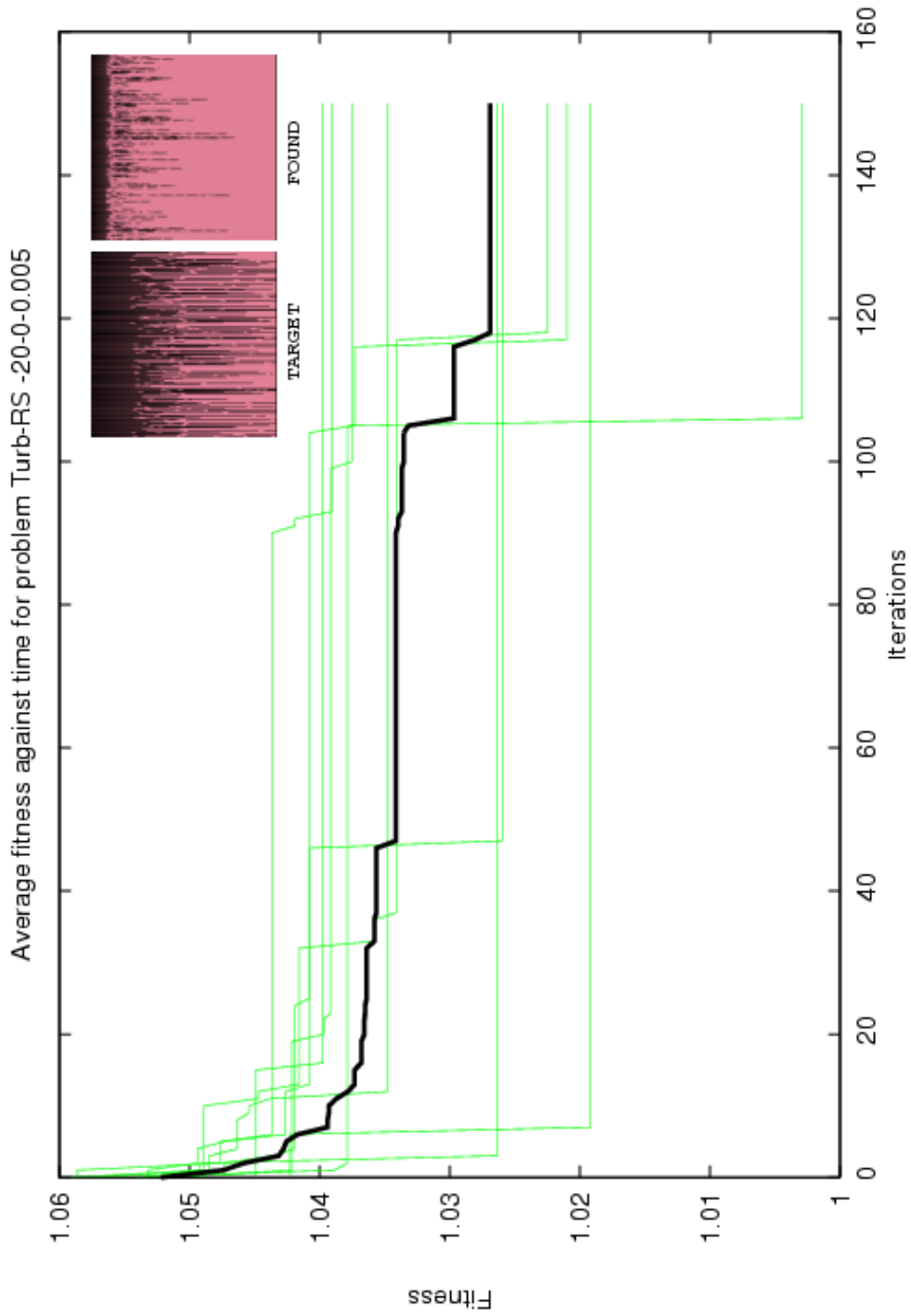


FIGURE 5.6: Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-20-0-0.005

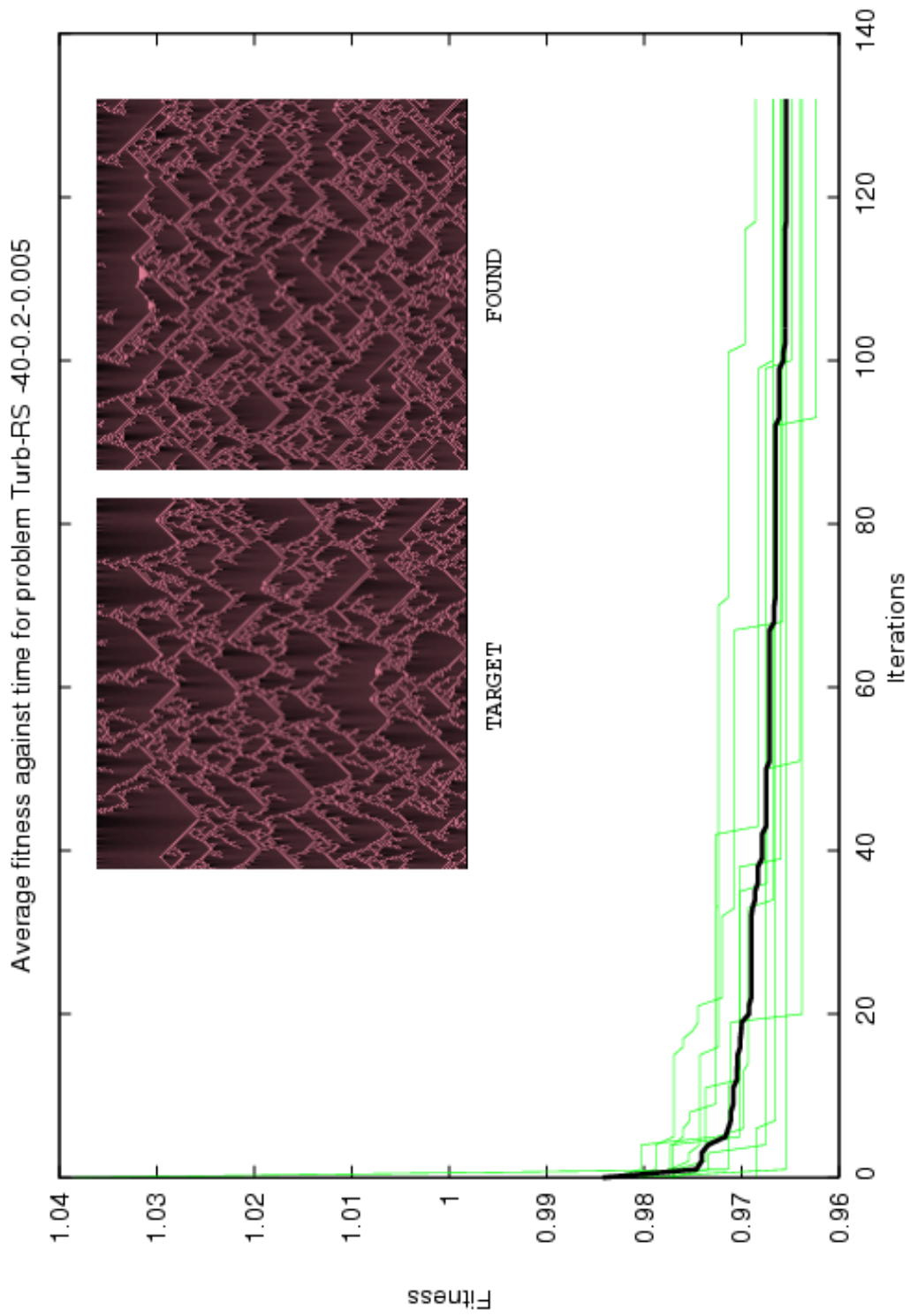


FIGURE 5.7: Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-40-0.2-0.005

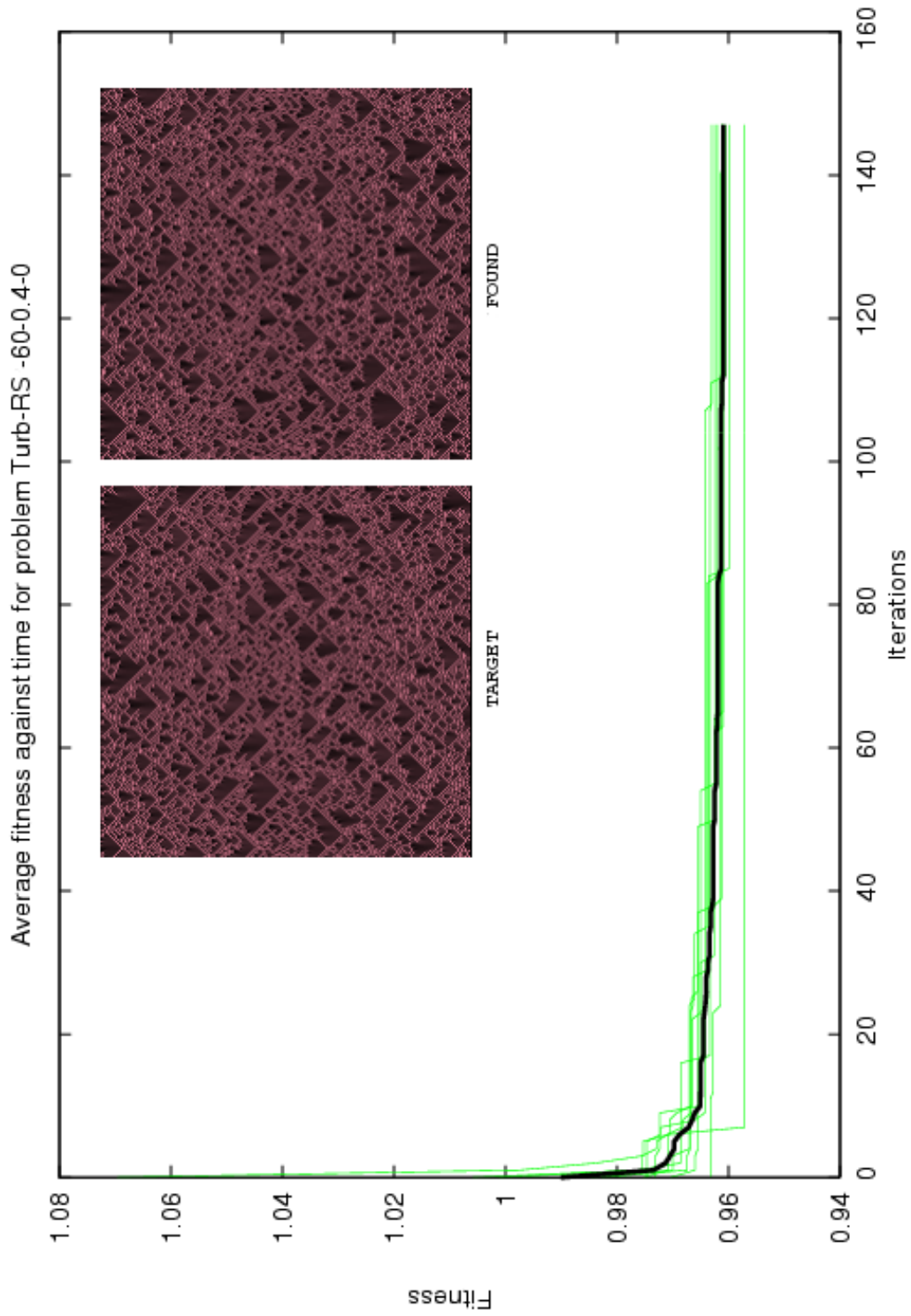


FIGURE 5.8: Graph of fitness against time, plus target and resultant behaviours for a random search of problem Turb-60-0.4-0

both in terms of visual similarity of the phenotype patterns and in terms of numerical fitness, with Random Search yielding higher (i.e. less similar) USM values in every case. This is not surprising – the probability of finding an optimal result by just randomly probing the search space is remote, and of course the larger the search space, the more insidious this problem becomes.

The results for Turb-40-0.2-0.005 and Turb-60-0.4-0 are rather closer in quality to those for the Genetic Algorithm, with closer USM values (though still worse in all but two cases), and better visual similarity. One possible explanation for this may be that the areas of the search space occupied by geneotypes that correspond to phenotypes similar to these two target patterns are, in comparison to those areas occupied by the other two targets, relatively large and flat. This type of landscape topology is much easier to search by any method, but the thinner, ‘spikier’ optima of targets Turb-20-0.8-0.005 and Turb-20-0-0.005 (as suggested by the much greater spread of evolutionary trajectories and resultant fitness values achieved) need a more comprehensive, intelligent search mechanism.

### 5.3.2 Hill Climbing

The results for the Hill Climbing algorithm are similar in character to those of the Random Search. As before, with targets Turb-20-0.8-0.005 and Turb-20-0-0.005, Hill Climbing does not match the quality of those for the Genetic Algorithm, certainly in terms of visual similarity, and to some extent, also in terms of numerical fitness.

Similarly, the results for Turb-40-0.2-0.005 and Turb-60-0.4-0 are rather closer in quality to those for the Genetic Algorithm, with closer USM values (though still, in general, not as good as those obtained with the GA).

In general, the results for Hill Climbing are comparable with those for Random Search. Neither of these local search methods, however, produce results quite as good as the Genetic Algorithm, due (particularly in the case of Turb-20-0.8-0.005 and Turb-20-0-0.005) to the uneven and narrow optima that characterise the fitness landscape for this, and most other complex problems.

The recombinative nature of the GA has been shown in numerous previous publi-

Target	Run	Generations	Fitness	Winning individual		
Turb-HC-20-0.8-0.005	1	150	0.98088	32.6	0.8	0.005
Turb-HC-20-0.8-0.005	2	150	0.99938	98.8	0.4	0.014
Turb-HC-20-0.8-0.005	3	150	0.99631	100.0	0.8	0.011
Turb-HC-20-0.8-0.005	4	150	0.99222	23.3	0.8	0.020
Turb-HC-20-0.8-0.005	5	150	0.98159	29.6	0.5	0.004
Turb-HC-20-0.8-0.005	6	150	0.98165	38.8	0.7	0.002
Turb-HC-20-0.8-0.005	7	150	0.99155	18.4	0.7	0.020
Turb-HC-20-0.8-0.005	8	150	0.99283	26.0	0.4	0.015
Turb-HC-20-0.8-0.005	9	150	0.99437	71.1	0.7	0.008
Turb-HC-20-0.8-0.005	10	150	0.96514	1.3	0.8	0.007
	$\mu$		0.98759			
	$c_v$		0.01025			
Turb-HC-20-0-0.005	1	132	1.01443	64.4	0	0.005
Turb-HC-20-0-0.005	2	132	1.03681	35.4	1.0	0.006
Turb-HC-20-0-0.005	3	132	1.00309	13.1	0	0.019
Turb-HC-20-0-0.005	4	132	1.03662	20.7	0.3	0.018
Turb-HC-20-0-0.005	5	132	1.02219	44.5	0	0.004
Turb-HC-20-0-0.005	6	132	1.01939	57.8	0	0.003
Turb-HC-20-0-0.005	7	132	1.03918	31.3	0.7	0.012
Turb-HC-20-0-0.005	8	132	1.03868	30.5	0.7	0.011
Turb-HC-20-0-0.005	9	132	1.03805	27.0	0.3	0.016
Turb-HC-20-0-0.005	10	132	1.01227	46.7	0	0.013
	$\mu$		1.02607			
	$c_v$		0.01339			
Turb-HC-40-0.2-0.005	1	147	0.95996	76.9	0.2	0.006
Turb-HC-40-0.2-0.005	2	147	0.96131	86.6	0.2	0.005
Turb-HC-40-0.2-0.005	3	147	0.97193	94.3	0.7	0
Turb-HC-40-0.2-0.005	4	147	0.97187	52.6	0.6	0.021
Turb-HC-40-0.2-0.005	5	147	0.97255	70.5	0.7	0.019
Turb-HC-40-0.2-0.005	6	147	0.97137	79.2	0.7	0.014
Turb-HC-40-0.2-0.005	7	147	0.96907	36.6	0.8	0.008
Turb-HC-40-0.2-0.005	8	147	0.96402	43.7	0.2	0.010
Turb-HC-40-0.2-0.005	9	147	0.97117	75.6	0.5	0.025
Turb-HC-40-0.2-0.005	10	147	0.97027	44.2	0.7	0.021
	$\mu$		0.96835			
	$c_v$		0.00475			
Turb-HC-60-0.4-0	1	145	0.96256	30.4	0.3	0.018
Turb-HC-60-0.4-0	2	145	0.96486	59.7	0.4	0.021
Turb-HC-60-0.4-0	3	145	0.96201	51.1	0.7	0.007
Turb-HC-60-0.4-0	4	145	0.96526	42.0	0.8	0.019
Turb-HC-60-0.4-0	5	145	0.96299	46.5	0.7	0.010
Turb-HC-60-0.4-0	6	145	0.96164	50.0	0.8	0.004
Turb-HC-60-0.4-0	7	145	0.96126	42.6	0.8	0.007
Turb-HC-60-0.4-0	8	145	0.96493	50.6	0.8	0.017
Turb-HC-60-0.4-0	9	145	0.95943	97.0	0.3	0.014
Turb-HC-60-0.4-0	10	145	0.96076	51.3	0.8	0
	$\mu$		0.96257			
	$c_v$		0.00195			

TABLE 5.3: Statistical analysis of Turbulence system hill climb

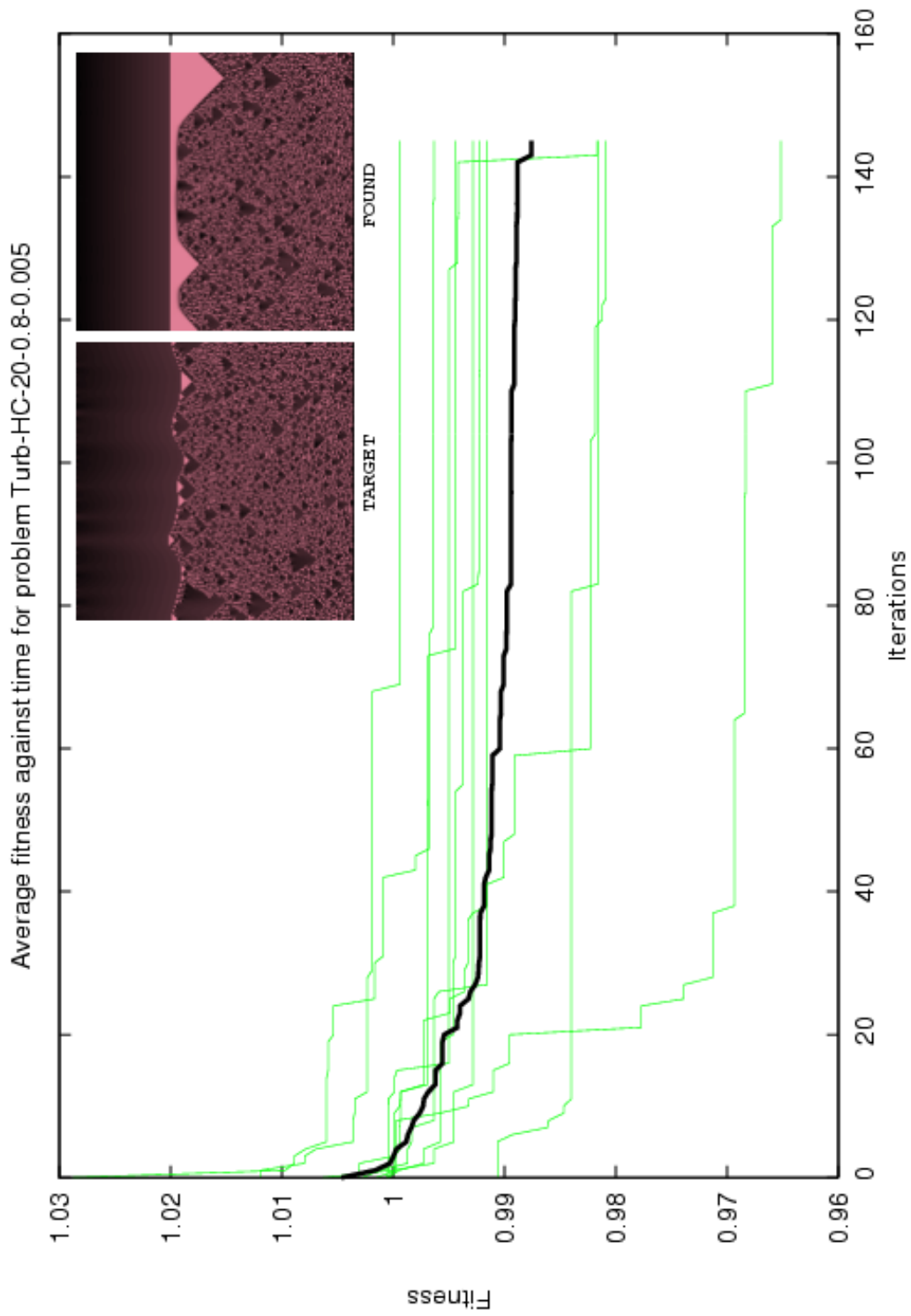


FIGURE 5.9: Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-20-0.8-0.005

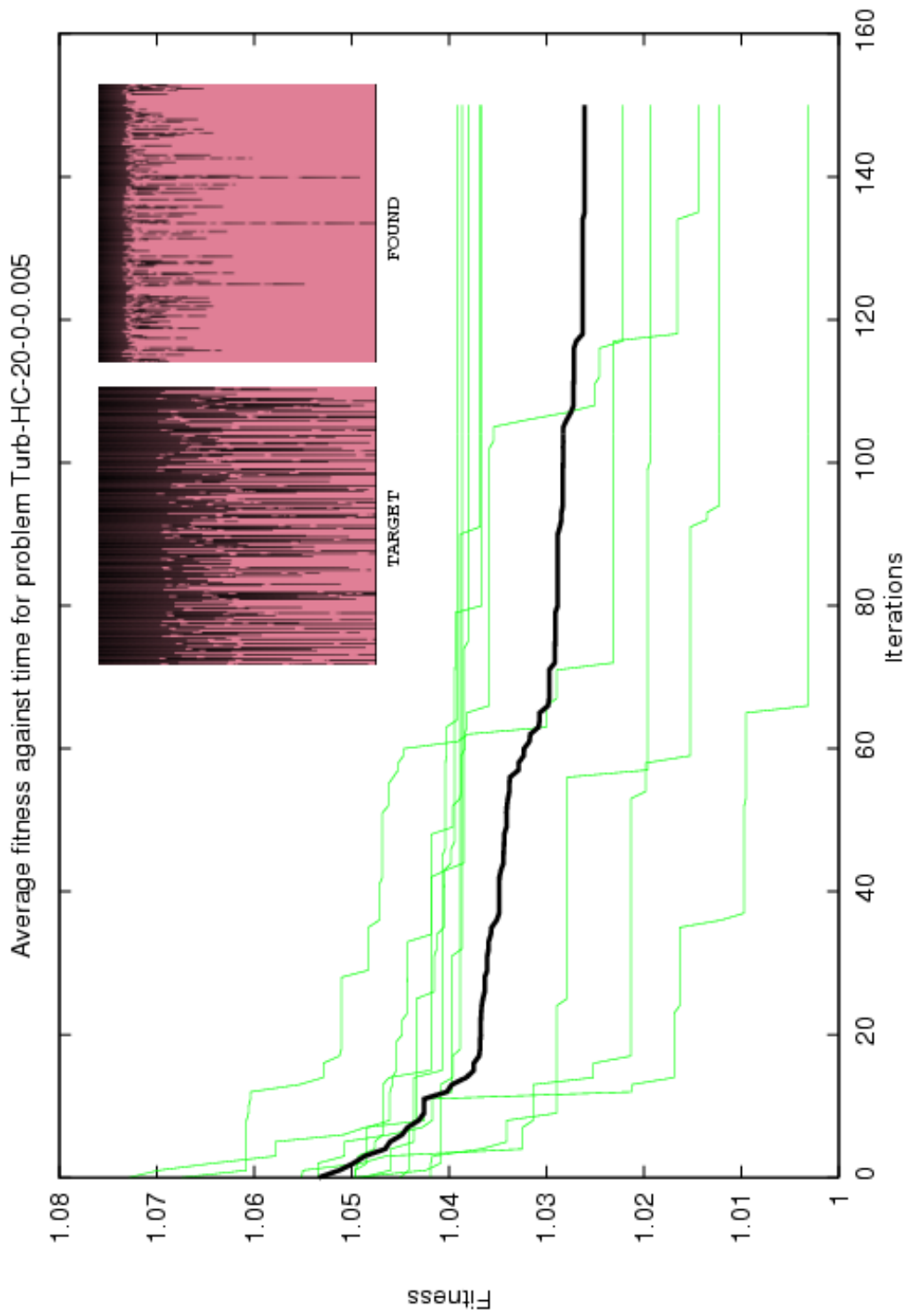


FIGURE 5.10: Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-20-0-0.005

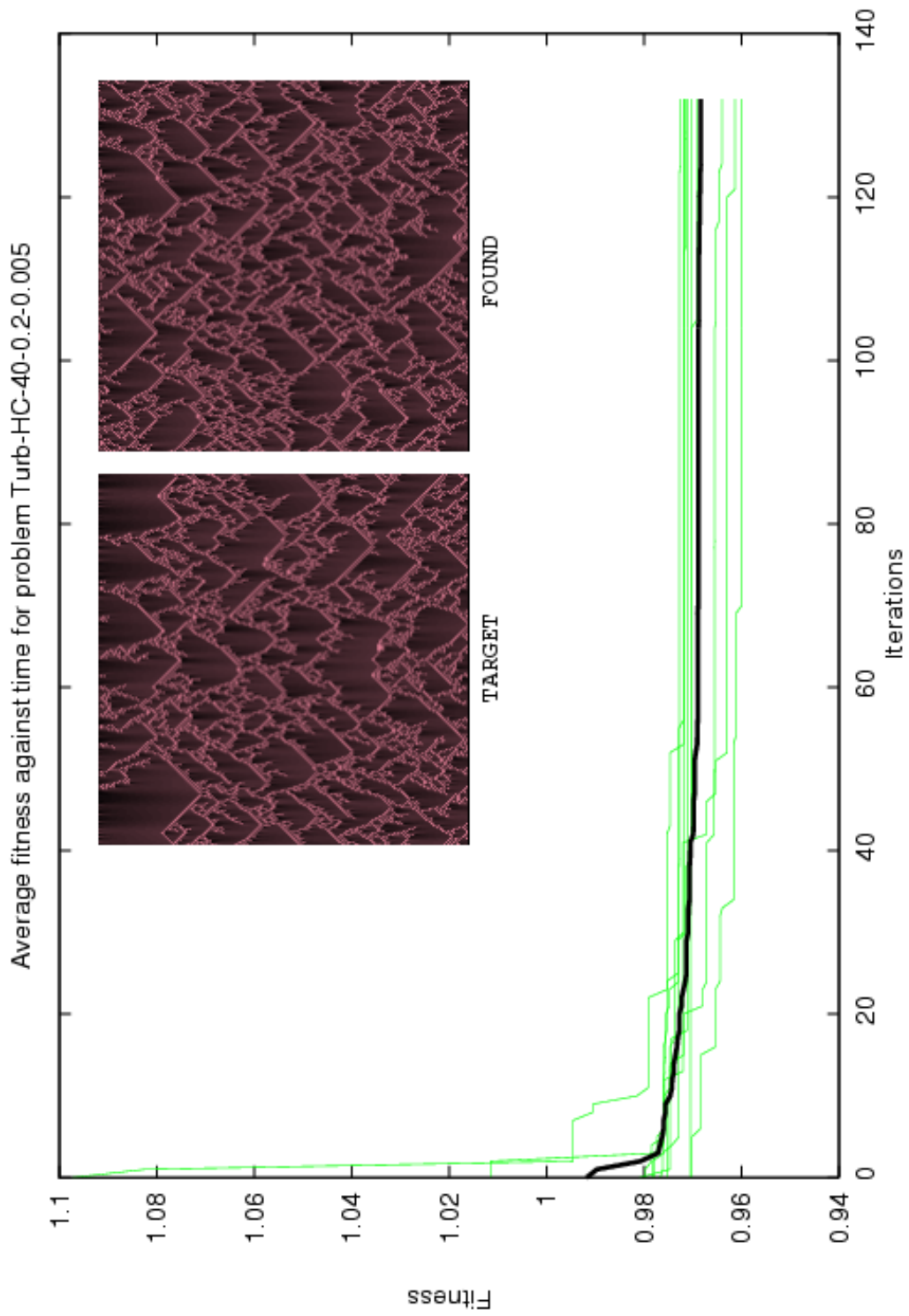


FIGURE 5.11: Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-40-0.2-0.005



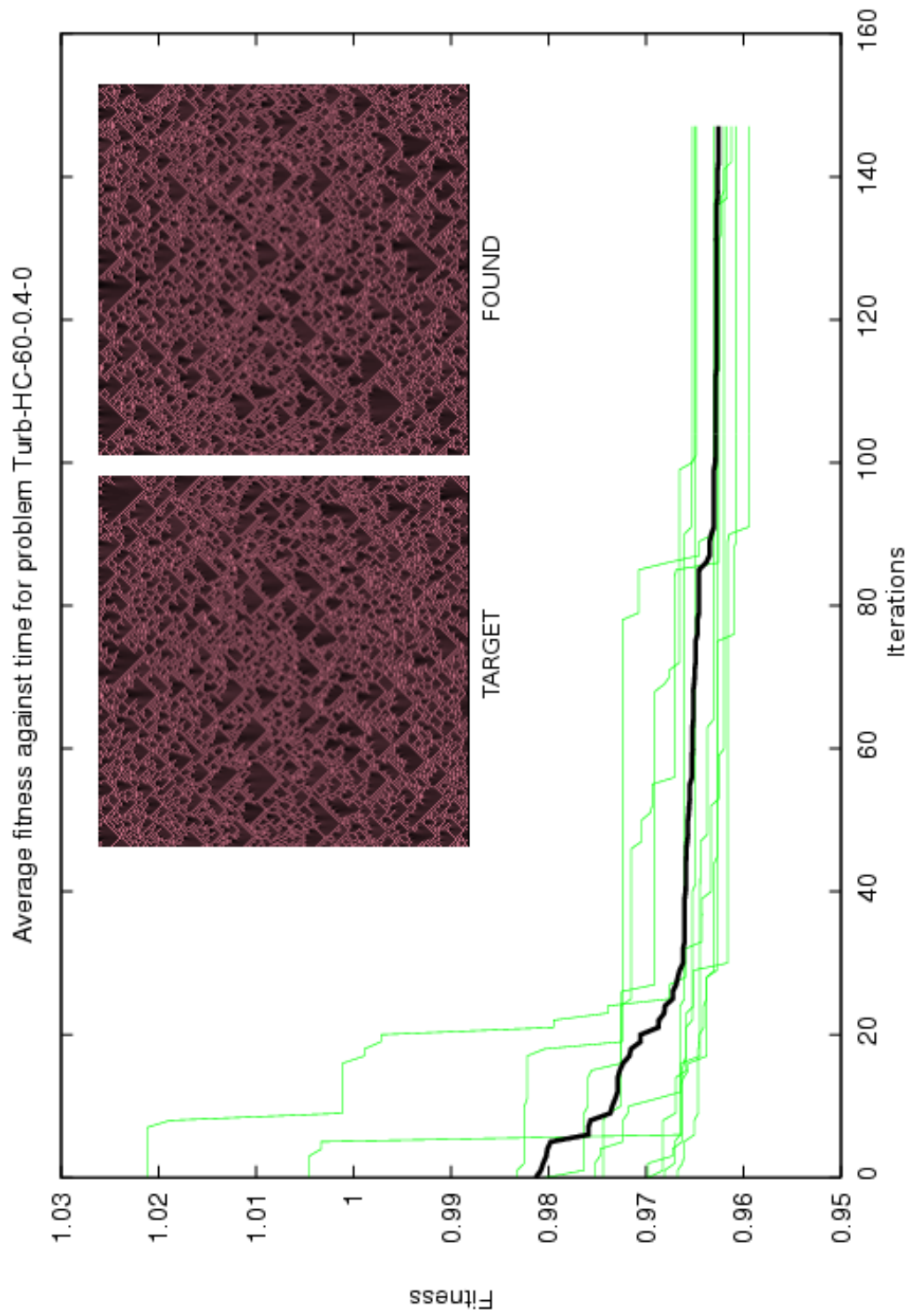


FIGURE 5.12: Graph of fitness against time, plus target and resultant behaviours for a hill climb of problem Turb-60-0.4-0

Experiment	GA vs RS	GA vs HC
Turb-20-0.8-0.005	13.559828	7.456722
Turb-20-0-0.005	6.082138	5.402137
Turb-40-0.2-0.005	3.297099	3.863920
Turb-60-0.4-0	10.064246	11.129315

TABLE 5.4: Statistical significance (t-value) analysis between the performance of Genetic Algorithm, Random Search and Hill Climbing methods.

cations [87, 102] to be effective in producing more successful offspring than a purely random search. The major advantage of a GA, though, is its population-based nature, and this is particular suited to complex problems with an innately parallel nature, as discussed previously. In a local search method such as Hill Climbing, the initial (random) ‘seed’ individual is of crucial importance – if it does not reside in the ‘trough’ of a good optimum, the algorithm is destined to failure. With a population-based mechanism, the probability of finding a fruitful area of the search space is much improved. A further advantage of a population-based mechanism, and hugely important when considering complex systems that may take a considerable time to compute, is that a single iteration of the algorithm, though comprising many evaluations, can be parallelised via the relevant mechanism in the Evolutionary Engine, described above in 3.2.

A simple T-Test was performed on the ten resultant fitnesses from each of the four target problems. Again, it is important to note that if the focus of this work was a true comparison of algorithmic efficiency, a more detailed comparison with more repeats and more sophisticated statistical methods should be employed, and indeed this may be the basis for future work, but to give a general picture, the t-values shown in table 5.4 are quite telling. With a degree of freedom of 18, significance levels range from 1.33 (alpha level of 0.2) to 4.97 (alpha level of 0.0001). If we choose the standard alpha value of 0.05, which corresponds to a significance level of 2.10, looking at the values in table 5.4, the difference between the GA results and those for both local search methods are clearly statistically significant in each of the four problems.

We have shown that the GA performs significantly better than random search and hill climbing in all the experiments performed on the *Turbulence* CA system. Given

these observations, coupled with the wealth of previous research that suggests local search methods are not appropriate for the optimisation of such complex problems, we are confident in selecting the Genetic Algorithm as the favoured search method to be used in the remaining complex search problems considered in this thesis.

## 5.4 A further assessment of the USM

In this section, the USM is evaluated further on a number of CA systems built using the Wolfram system described in section 4.1.

### 5.4.1 Problem description

In [86], a GA was used to evolve a non-uniform CA – a development of the CA paradigm where each cell in the lattice does not use the same rule set – this makes the system considerably more complex. We expand our system in a similar manner, such that the CA can be partitioned to use a number of different rules within a single model; for example, in a CA whose grid is bounded to a size of 100 x 100, columns 0-50 may use Wolfram Rule 30, but columns 51-100 may use Rule 145. This gives the USM the much greater challenge of having to capture two distinct behaviours and, as will be shown later on, demonstrates some particularly interesting features of the metric’s abilities.

Three groups of target patterns were defined using this ‘meta-automaton’ system. In the first set, all columns use the same rule. The ten target patterns produced by these automata are shown in figure 5.13. For the second group of target patterns, the spacial dynamics were divided in two. That is, given two random rules chosen from the ‘pool’ of 256 rules, the first 50 cells are associated with one rule and the remaining 50 with the other. These nine targets are shown in figure 5.14. In order to provide the USM/GA search system with an even more challenging data set, the spacial dynamics were further divided into four for the last group of target patterns. In this case, cells were divided into groups of 25 consecutive cells and a randomly selected rule was associated to each group. These four patterns are shown in figure 5.15.

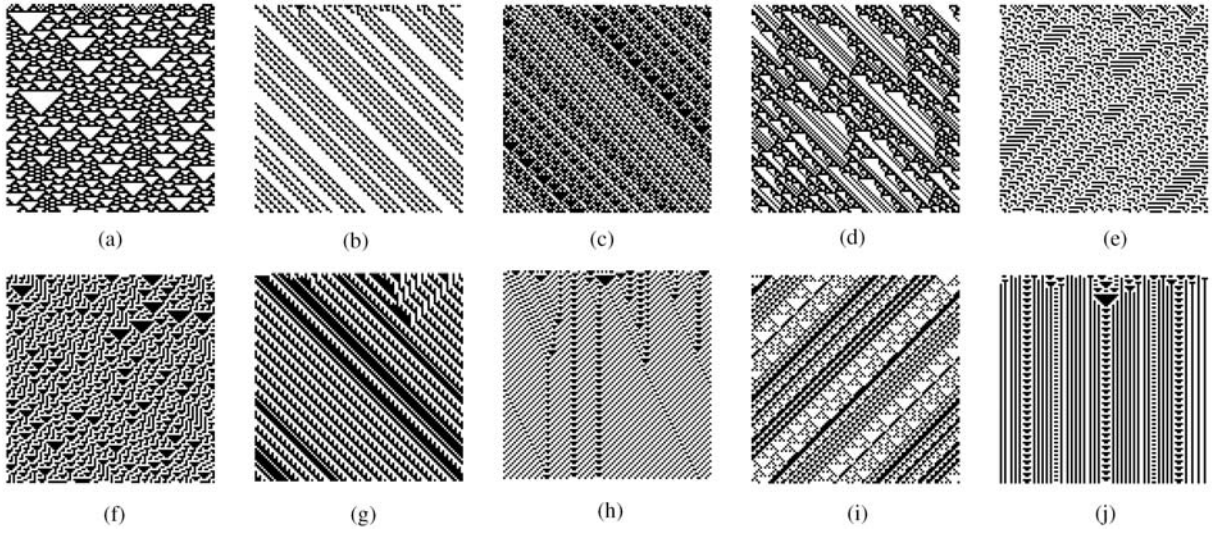


FIGURE 5.13: Target patterns for meta-automaton group A (one rule per model)

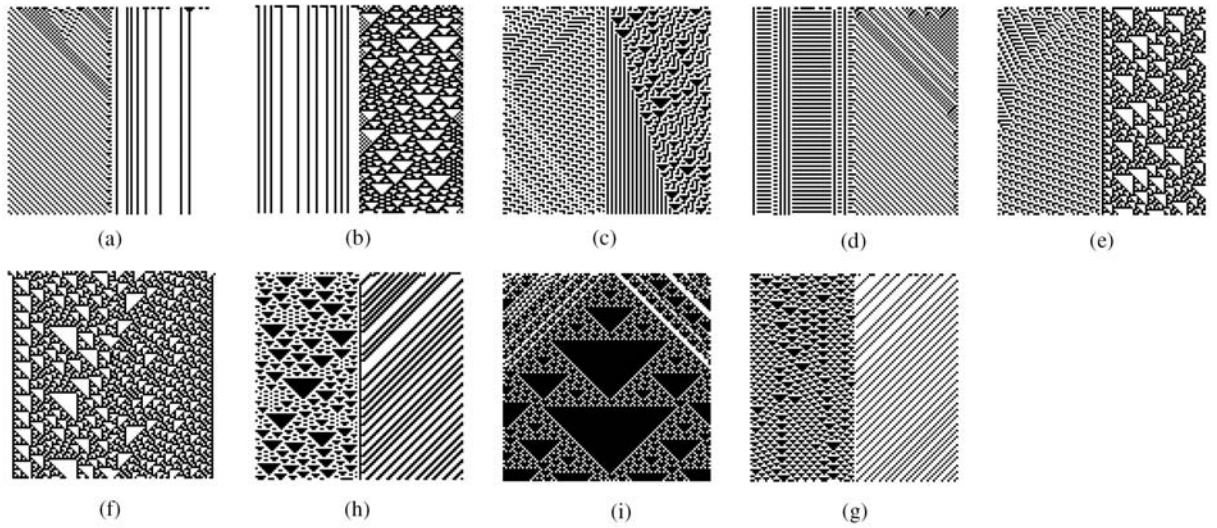


FIGURE 5.14: Target patterns for meta-automaton group B (two rules per model)

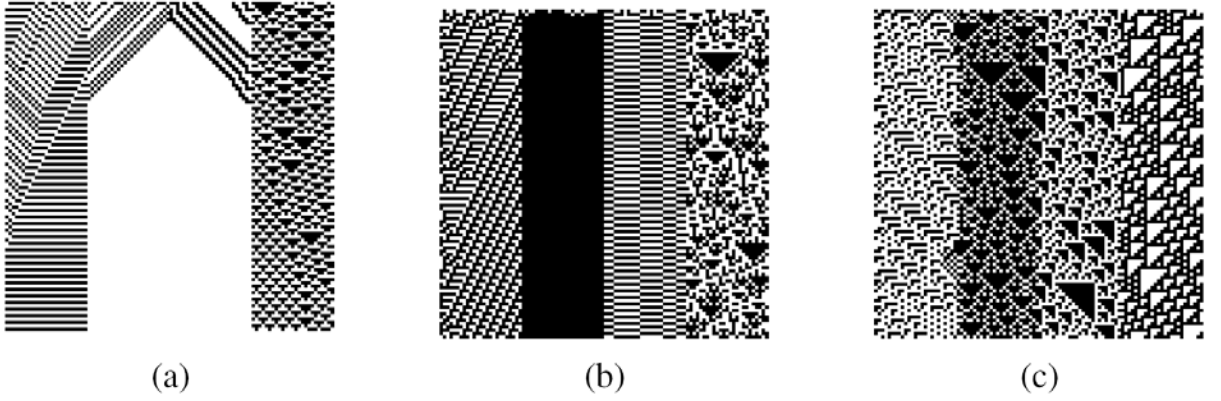


FIGURE 5.15: Target patterns for meta-automaton group C (four rules per model)

#### 5.4.2 Results

A GA using the USM as fitness function was run for 100 generations on each target. Table 5.5 shows the target and evolved rule sets, the associated USM value, as well as an indication of visual similarity. As shown, five out of ten experiments evolved the expected rule for the first data set. However, if we further analyse the remaining results we see that in most of the cases where the expected rule was not achieved, the evolved rule often results either in a mirror or otherwise similar image. For instance, as figure 5.16(a, b) shows, the evolved target patterns found for target A2 and A4 are mirror images. Moreover, as depicted in figure 5.17(a), it is clear that the diagonal black strips appearing in target A9 were well-captured by the USM. It can be argued, therefore, that certain similarities between the target pattern and the designoid were found in most cases. Contrarily, in the case of targets A6 and A7, no similarities at all appear (despite an encouraging USM value for A6 in particular).

In the case of the second data set, equivalent rules, mirrors and close similarities are also found. None of the results have reached exactly the correct rules, however, five results out of ten are visually similar – three evolved patterns produce mirror images, and two produce important features appearing in the target patterns. Figure 5.16(c, d, e) show that a target rule plus an equivalent rule were found in the case of the mirrored B1, B2 and B7. On the other hand, figure 5.17(b, c) shows that in the

Target ID	Target ruleset	Evolved ruleset	USM value	Visual similarity
A1	[122]	[122]	0.993958124	Correct
A2	[148]	[6]	1.042744644	Mirror
A3	[181]	[181]	0.984504855	Correct
A4	[120]	[106]	0.983119009	Mirror
A5	[97]	[97]	0.985429776	Correct
A6	[135]	[195]	0.976343879	None
A7	[229]	[195]	1.048922986	None
A8	[131]	[131]	1.00218998	Correct
A9	[154]	[169]	0.987069886	Captured
A10	[133]	[133]	0.950053315	Correct
B1	[177 132]	[164 177]	0.818578016	Mirror
B2	[68 122]	[122 100]	0.885830497	Mirror
B3	[65 135]	[215 146]	0.948304844	None
B4	[5 57]	[115 192]	0.870995252	None
B5	[25 60]	[26 125]	0.96081944	None
B6	[60 102]	[183 20]	0.964207074	None
B7	[147 2]	[130 147]	0.905361748	Mirror
B8	[129 46]	[126 16]	0.958283213	Captured
B9	[167 180]	[91 167]	0.993560531	Captured
C1	[49 34 84 147]	[73 141 188 230]	0.907788419	Captured
C2	[61 251 23 165]	[38 140 105 234]	0.917228868	Captured
C3	[41 183 195 110]	[61 120 146 196]	0.940763235	Captured

TABLE 5.5: Target and evolved rule sets for the meta-automaton system

case of B8, the white-on-black triangular pattern on the left of the image becomes a very similar, but black-on-white pattern in the evolved version.

In the case of the third data set, mirrors were more difficult to produce and none of the evolved patterns have reached the target rules. However, a visual analysis of the obtained patterns supports the idea that some relevant features were captured from the target images. For example, in figure 5.18(a) it is particularly interesting to note that a pair of rules for producing the central inverted V-shape were discovered (but with reversed colouring and in a different position). Moreover, in figure 5.18(b), an equivalent rule for the second strip was discovered at the fourth position in the evolved pattern, and the chaotic behaviour of the last strip is represented in the third position of the evolved version. Finally, in figure 5.18(c), a similar effect of colour inversion is observable between the second and third strip of the target and evolved

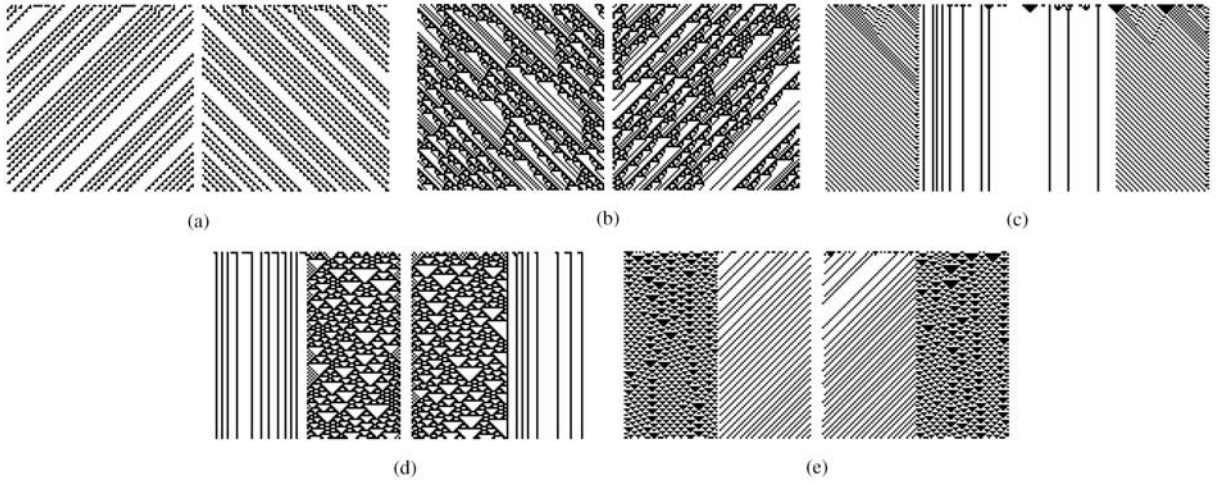


FIGURE 5.16: Examples of mirror images evolved for meta-automaton targets (target, left; evolved, right): (a) A2, (b) A4, (c) B1, (d) B2, (e) B7

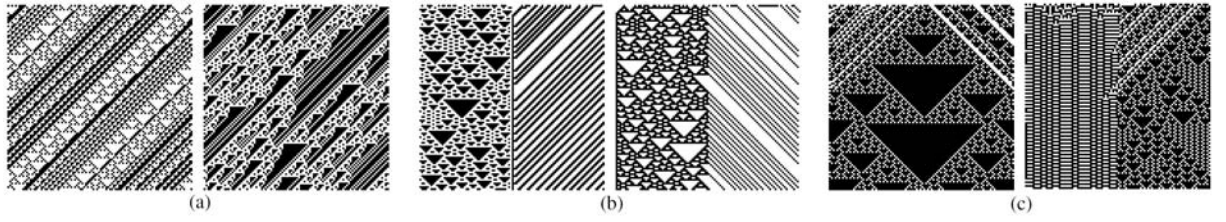


FIGURE 5.17: Examples of other similarities captured in evolved meta-automaton targets (target, left; evolved, right): (a) A9, (b) B8, (c) B9

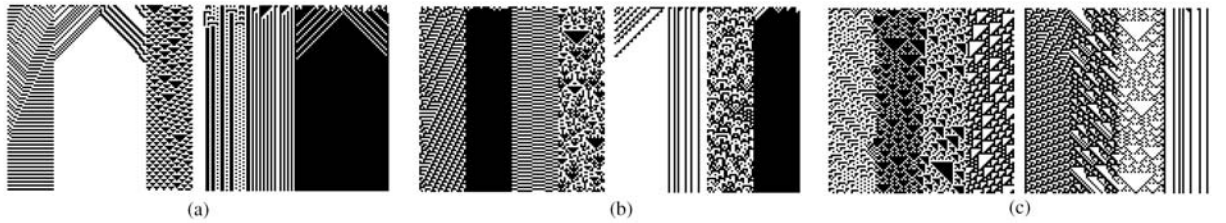


FIGURE 5.18: Target and evolved patterns for meta-automaton group C (four rules) (target, left; evolved, right): (a) C1, (b) C2, (c) C3

patterns respectively.

### 5.4.3 Outcomes

Even in those results where the exact rules have not been found, the nature of the rules used in this ‘meta-automaton’ mean that a number of different rules can have very similar spacio-temporal behaviour. Hence, as seen in the results for the *Turbulence* model earlier in this chapter, a significantly different genotype can, in fact, result in a similar phenotype – further illustration of the complex, non-linear nature of the genotype–phenotype–fitness mapping in these systems. It is evident from these experiments that, although the USM works well in many cases, it has a number of shortcomings. As illustrated by the results above, one of the most obvious is its blindness to negative images (intuitively, using the description of conditional Kolmogorov complexity described previously, the amount of information needed to produce, for example, a segment of black pixels, given a segment of white pixels is equal to that needed to produce a segment of white pixels given a segment of black pixels). Similarly, the USM does not differentiate between mirror images. These interesting insights suggest that although useful in many cases, the USM may not be as universally appropriate as at first thought. Indeed, in the next chapter, the USM is not found to be an appropriate fitness method, and an alternative metric is proposed and investigated.

## 5.5 Conclusions

The Evolutionary Engine’s ability to successfully coerce a cellular automaton-based system into a pre-defined, target behaviour has been demonstrated, using a fitness function based on the Universal Similarity Metric. Furthermore, the results obtained from the genetic algorithm were compared to two standard non-evolutionary alternatives, and the argument made that the evolutionary methodology is better suited to this area of complex system design. The USM’s ability to evolve target behaviours was further verified by using a number of other CA-based systems; although the metric scaled up to this new problem family, a number of interesting shortcomings were



identified, suggesting the method may not be universally successful.

Having proved the concept of the Evolutionary Engine using relatively simple CA models, we move on now to look at a more complex model that closely simulates a real physical process.

## CHAPTER 6

# Evolutionary design of nanostructures

From the abstraction of CA-based models, we move now to consider, for the first time, the evolutionary design of target behaviours on a real-world problem derived from the nanosciences. In this chapter, the Evolutionary Engine is applied to a Monte Carlo model that simulates the self-organisation processes of passivated gold nanoparticles when spin cast onto a silicon substrate. The engine is used, first with the USM, then in conjunction with an innovative fitness function based on Minkowski functionals, to tune the operation of this system towards a target behaviour (arrangement of particles), represented by a graphical pattern.

The process of the adsorption of gold nanoparticles not only produces an impressive range of patterns, but has previously been proved [98, 80] to be remarkably well-described by a relatively simple Monte Carlo algorithm. This application takes the work of this thesis much closer to the design of real, physical systems, as well as providing an important link between simulation and experiment in the study of self-organising nanostructured systems. The potential of this work to be extended to the construction of nanoscale components as part of an ‘unconventional computing’ system, such as that suggested in [126] is particularly relevant in today’s research environment, and indeed, there is a clear analogy between the behaviour of such a system and the origins of biological life – the development of both areas relies on the self-assembly of simple particles into an arrangement whereby some form of complex behaviour emerges.

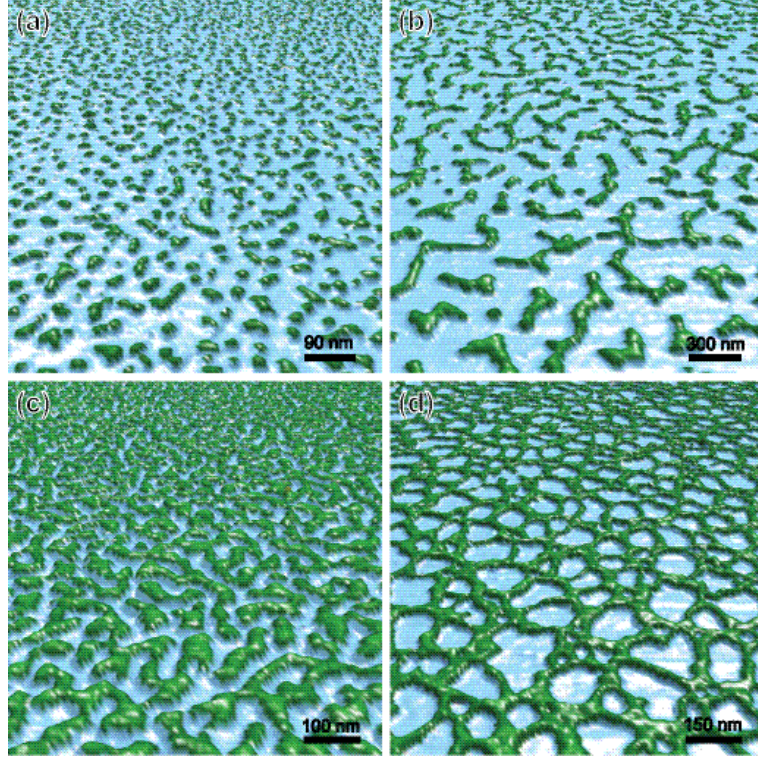


FIGURE 6.1: 3D atomic force microscope images demonstrating four types of commonly observed nanostructured patterns. Imaging by Christopher Martin/Philip Moriarty (University of Nottingham)

## 6.1 Nanostructures

Nanoscience is the study of extremely small-scale systems. A nanometre is  $10^{-9}m$ , which equates to roughly  $\frac{1}{70000}$  of the width of a human hair. Systems that operate on this scale are of obvious interest in the seemingly unstoppable quest towards the further miniaturisation of technology, specifically to fields such as computing and medicine. Among the many avenues of research in this new and quickly developing field of science, is the study of nanoscale self-organising systems. Governed by a number of experimental parameters, so-called ‘nanoparticles’ can organise themselves into a large variety of patterns. In time, these patterns may be able to encode particular functions [126], and this is therefore a particularly important area of research. Figure 6.1 shows a selection of atomic force microscope images of such patterns.

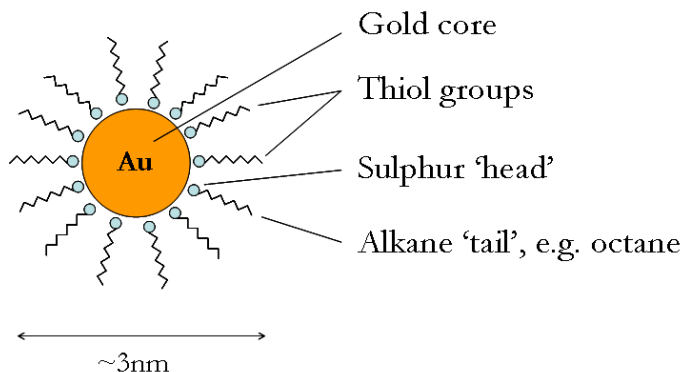


FIGURE 6.2: Diagrammatic representation of the construction of a Thiol-passivated Gold nanoparticle

### 6.1.1 Physical background

When deposited onto a solid substrate, colloidal nanoparticles (that is to say, a number of nanoparticles dispersed evenly throughout a solvent) have been found to self-organise into a variety of complex patterns [39, 98, 88, 80, 89, 6, 9] principally driven in many cases by the evaporation of the solvent. The system of interest in this chapter, namely, gold nanoparticles dispersed in toluene deposited onto a silicon substrate, has been described at length in a number of earlier publications [88, 80, 9]. Figure 6.3 shows a selection of the different types of morphologies that can be obtained. The nature of the pattern depends on a number of factors including nanoparticle concentration, the nature of the solvent and substrate, and the length of the thiol groups (see figure 6.2) that surround each gold particle. Understanding the physical processes that govern the self-organisation of patterns like those shown in figure 6.3 is an area of particularly intensive research, and one in which simulations and experiments play a vital role.

### 6.1.2 Simulation

The system built, developed and described by the authors of [80] to simulate the processes described above is based on a two-dimensional Monte Carlo model introduced in [98]. The solvent is represented as an array of cells on a square grid, each

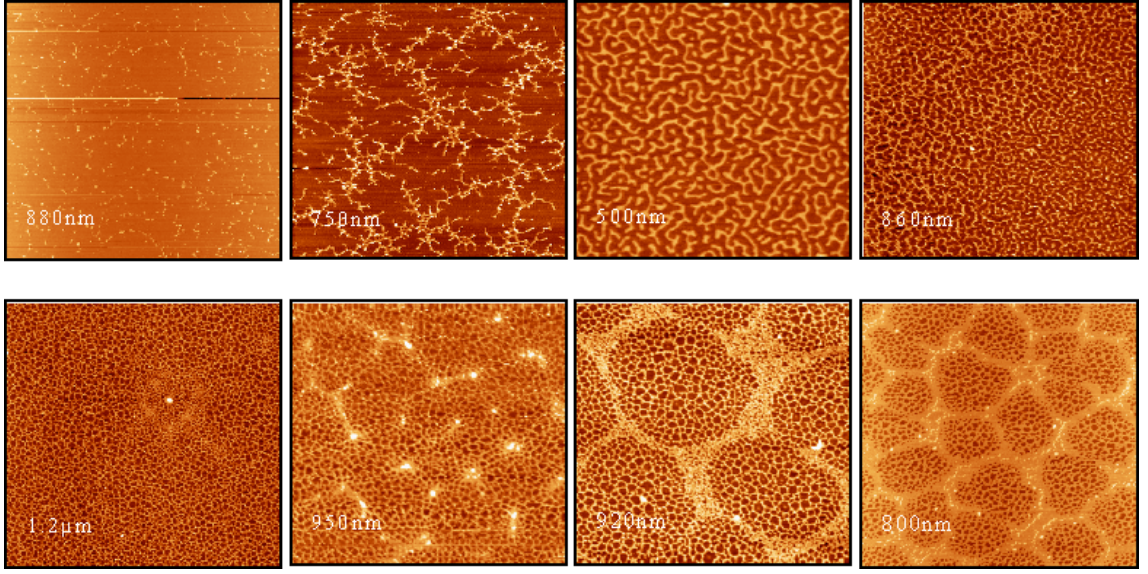


FIGURE 6.3: Example morphologies obtained through the spin casting of Thiol-passivated Gold nanoparticles onto a silicon substrate

of which represents  $1nm^2$ , and can have a value of either 1 or 0 to represent liquid or vapour respectively. Each gold nanoparticle occupies an area of  $3 \times 3$  cells, and liquid is excluded from the sites where a particle is present. The simulation proceeds by two processes: the evaporation (and recondensation) of solvent, and the random walks of nanoparticles. The simulation is subject to a number of parameters that determine the nature of the pattern formed as output. The stochastic nature of the model means that the same parameter set is unlikely to produce exactly the same pattern more than once. However, the degree of stochasticity is such that the visual characteristics of the pattern generated by a given parameter set all have similar, albeit non-identical, features.

As with the cellular automaton-based systems studied in the previous chapter, the nanosystem simulator studied here has a number of numerical input parameters which determine the nature of the pattern formed as output (just as the experimental laboratory conditions do). The simulator is coupled to the Evolutionary Engine in order to tune the input parameters so as to obtain a pattern as similar as possible to a pre-defined target morphology. The simulator takes four parameters as input; as before, we quote lower and upper bounds, as well as an indication of sensitivity for

each parameter. Parameter  $MR$  [5, 50, 1] governs the mobility ratio of the particles, in effect, an abstraction of the particles' interaction energies, determining how fast the particles move compared to the evaporation of the solvent. The *coverage* parameter [0.05, 0.29, 0.01] is simply the concentration of nanoparticles dispersed in the solvent and  $kT$  [0.5, 2, 0.1] a measure of the energy (temperature) of the system.  $EL$  [1, 4, 0.1] defines the attraction strength between the solvent and solvent, or between the solvent and particle.

The output of the simulator is a greyscale image representing the pattern formed by the self-organisation of the nanoparticles where particles themselves are represented as grey pixels, whilst residual solvent is represented by black pixels and the white areas represent uncovered areas of substrate. We consider 'covered area' to be those areas covered by *either* particle or solvent, and can therefore threshold the output images into a binary grid where each cell is either 'covered' (black) or not (white).

The accuracy, lower and upper bounds of each parameter are intuitively derived from the physicists' understanding of the system. It is in the interests of the optimisation process to have as small (yet phenotypically diverse) a search space as possible, and indeed the upper bound of one of the parameters to the simulator, *coverage*, that specifies the concentration of nanoparticles in the toluene solvent, can affect such a change. We can place a revised, target-specific upper bound on this parameter by measuring the covered area of the target pattern. Note that it is only an upper bound (i.e. not an exact value), because the amount of residual solvent (which we also consider as 'covered area') that will be left behind is not known and can vary considerably.

Examples of the morphologies obtainable through use of this simulator, and the parameters used to generate them can be found in the appendix (A.2). Four specific examples are shown in table 6.1.

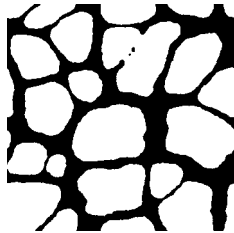
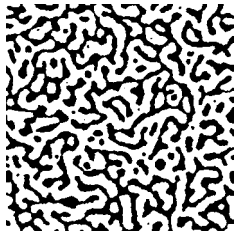
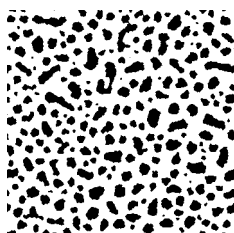
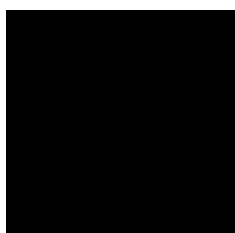
<p style="text-align: center;">“A”</p>  <p style="text-align: center;">Parameters: 20, 0.29, 1, 3 Time: 941s Minkowski: 360057, 17768, -7</p>	<p style="text-align: center;">“B”</p>  <p style="text-align: center;">Parameters: 5, 0.29, 1, 2 Time: 589s Minkowski: 428606, 52828, 56</p>
<p style="text-align: center;">“C”</p>  <p style="text-align: center;">Parameters: 35, 0.21, 2, 3 Time: 978s Minkowski: 332788, 44734, 265</p>	<p style="text-align: center;">“D”</p>  <p style="text-align: center;">Parameters: 5, 0.05, 1, 4 Time: 538s Minkowski: 1048576, 4096, 1</p>

TABLE 6.1: Example patterns from the nano system simulator. The patterns are shown along with the parameters used to generate them, the time taken for the simulation to be produced (on an Intel Pentium 4 CPU running at 2.40GHz), and their Minkowski functionals (which are explained later).

## 6.2 Fitness

In order to guide the Monte Carlo simulator into producing a particular morphology, a method of measuring similarity between self-organised patterns (target *versus* evolved) must be used, just as the USM was used in the previous chapter in the context of CA-based systems.

### 6.2.1 The Universal Similarity Metric

Given the success of the USM in evolving target behaviour in the CA system, as presented in chapter 5, initial investigations were concerned with verifying whether the USM would be an effective method for directing a search of this new system.

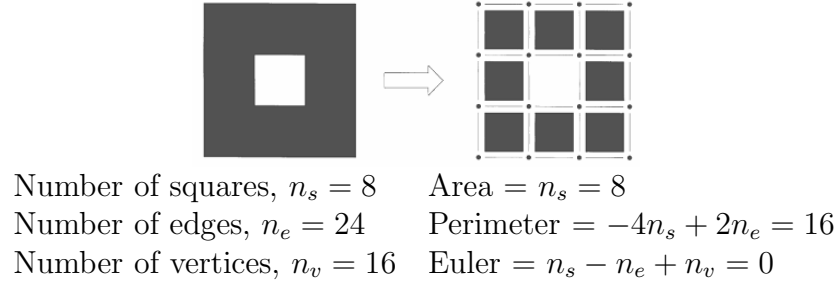
A cluster tree was produced, as described previously in chapter 4. As with the CA-based system, it is important that the dataset used for this cluster analysis is representative of the whole search space, whilst still being a manageable size. Hence, the search space is systematically sampled as we did with the *Turbulence* system before. For the four genes, *MR* [5, 50], *coverage* [0.05, 0.29], *kT* [0.5, 2] and *EL* [1, 4], the number of bins,  $b$ , is set to four, giving a dataset size of  $4^4 = 256$  points. The entire dataset is listed in appendix A. As before, every pattern of the dataset was compared for similarity to every other, and a square similarity matrix obtained. Figure 6.4 shows the corresponding hierarchical cluster tree. It is clear that in this problem domain, the USM's ability to effectively classify similar patterns is poorer than expected. Although the darker patterns (clusters D-G) are set apart in the tree, it is particularly concerning to see that a number of clusters representing visually similar patterns (such as H, K and M, or A and I, or B and L, or D and G) are not placed together in the tree. Moreover, the USM does not seem to be able to effectively differentiate between patterns with a high degree of connectivity. That is to say that patterns that consist of a number of isolated and unconnected regions (such as those individuals represented by clusters M, K and H) are not sufficiently partitioned from those individuals (such as A, B, L and I) that are characterised by more connected 'worm-like' or labyrinthine structures. This, coupled with the shortcomings identified in section 5.4 suggests that a more introspective, 'geometrically aware' fitness function



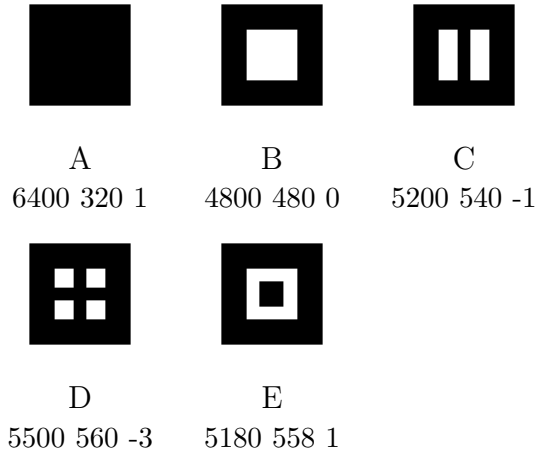
is required.

### 6.2.2 Minkowski functionals

The authors of [80] characterised certain aspects of similar simulation patterns using the Minkowski functionals. We develop this work, constructing a similarity measure based on these measures. The Minkowski functionals [84] characterise a binary pattern in terms of area, perimeter and Euler characteristic (a measure of connectivity). They have been recently used in fields such as tile design [119] and vesicle simulation [78]. A binary pattern can be sub-divided into pixel squares and the three functionals calculated as follows:



In this way, precise geometrical characteristics can be extracted. These characteristics are explained further by illustration below:



Graphic A shows a simple square of 80x80, from which the area and perimeter characteristics are trivial; for all simple polygons (that is, without holes), the Euler

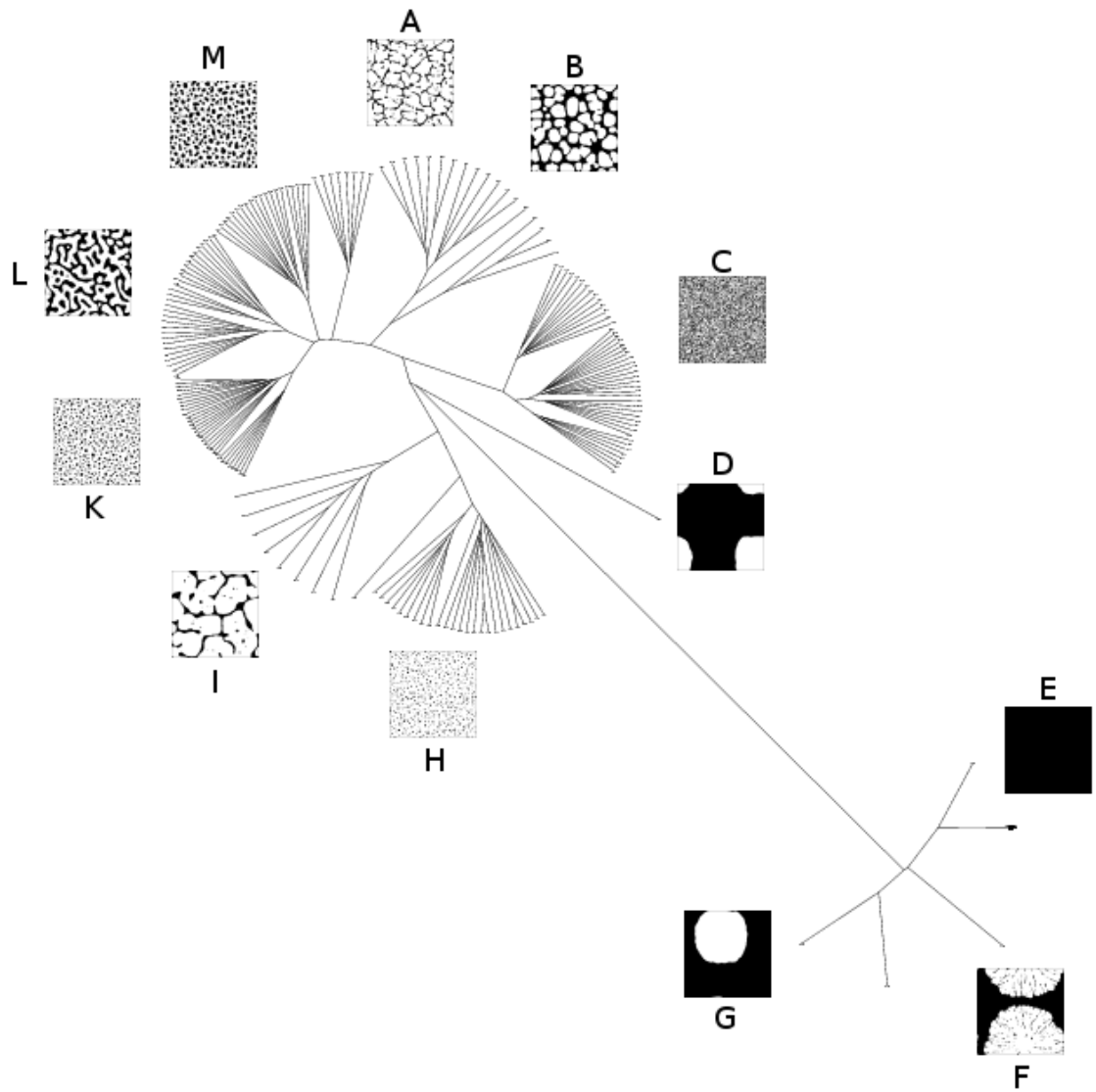


FIGURE 6.4: Hierarchical clustering tree showing USM classification of the nano system dataset

characteristic equals one. For figures *with* holes, the Euler characteristic will be less by the number of holes present, as illustrated by Graphic B (which also has a greater perimeter, due to the newly introduced inside edge). Adding a further hole (Graphic C) has the expected effect (again also increasing the perimeter), and furthermore, Graphic D. Note that adding a second body *within* the hole (Graphic E) causes the Euler characteristic to increment again, although this graphic is differentiated from A in terms of both area and perimeter.

As a further, more problem-specific illustration and explanation of this method, table 6.1 shows four example patterns along with their corresponding Minkowski values. Pattern A is a highly connected, cellular structure. The large scale of these ‘cells’ means the total perimeter of the pattern is relatively low (and much lower than the similarly connected, but much smaller length-scale pattern B). The small, negative Euler characteristic is representative of the fact that the structure is almost entirely connected, with a small number of ‘holes’. Pattern B is also quite highly connected (but not cellular), hence the relatively small Euler number; the perimeter is much higher than that for pattern A due to its smaller length scale, as discussed above. Pattern C has a high Euler number, and a reasonably high perimeter, as the pattern is a highly disconnected collection of small ‘islands’. Pattern D clearly has the maximum area value and a Euler number of 1, as the simulation space is completely covered.

It is important to note that for these nanostructured patterns, the scale of each functional varies considerably. For example, the area functional takes values in the region of 200,000 - 500,000; perimeter is a factor of ten smaller at between 10,000 and 80,000; with the Euler characteristic a further factor of 100 smaller with values between 0 and 700. Using simply the root mean squared of the sum of each error value (that is to say, the Euclidian distance between the target functional and that produced by the evolved pattern) would introduce a false weighting into the fitness function in favour of the numerically larger functionals. We are therefore faced with two options – either, the error value for each functional is normalised and then combined into a single similarity value to be minimised by the GA objective function, or each error value is optimised independently in a multi-objective setting. Although simpler, the

use of normalisation factors and an aggregate function will inevitably introduce noise into the fitness function, hence we have opted to employ a multi-objective approach for the work which follows. As explained in chapter 3, for multi-objective problems, the Evolutionary Engine employs the popular Non-dominated Sorting Genetic Algorithm (NSGA-II) invented by Deb [31] – one of the most popular multi-objective genetic algorithm implementations used in industry today.

Each of the three Minkowski error measures (the absolute difference between the relevant target and evolved functional) are allocated as a separate fitness component (equation 6.1 where  $a$  = area,  $p$  = perimeter,  $e$  = Euler,  $t$  = target,  $v$  = evolved). At the completion of the algorithm, the user can then choose which individual from the final Pareto front to accept as the best solution.

$$F_i = \langle |a_t^i - a_v^i|, |p_t^i - p_v^i|, |e_t^i - e_v^i| \rangle \quad (6.1)$$

### 6.2.3 Stopping condition

In standard single objective optimisation, stopping conditions are relatively simple to construct. Common techniques include a simple cap at a predefined number of generations, a predefined fitness value that, when reached, signals termination, or stopping the algorithm after a certain number of iterations showing no improvement in average fitness (as implemented in the experiments presented in chapter 5). The first technique mentioned above, that is, imposing a set number of generations over which the search should run, has a number of major disadvantages, not least that it is not suitable for performance comparison. Stopping the algorithm when further improvement looks unlikely (and a fixed number of unimproved generations is a good estimate of this) is a much better option, enabling the efficiency of a search to be measured in terms of time (generations). The very nature of a multi-objective algorithm, however, considerably complicates the definition of a ‘non-improvement’ measure, as there is no single fitness to measure. In the results presented below, we use the technique proposed in [106]: The maximum crowding distance of all members of the Pareto set,  $d_l$ , is measured each generation, and a history of length  $L$  maintained. We stop the algorithm if the coefficient of deviation of this set of distances is

less than or equal to a defined boundary,  $\delta$ :

$$\frac{\sqrt{\frac{1}{L} \sum_{l=1}^L (d_l - \bar{d}_L)^2}}{\bar{d}_L} \leq \delta \quad (6.2)$$

In the experiments presented below, it was found, by preliminary experimentation, that  $L = 10$  and  $\delta = 0.2$  are suitable parameters.

The simulator is reasonably compute and time intensive (as illustrated by table 6.1, a single run can take almost twenty minutes to run). Therefore, the distributed evaluation option provided by the evolutionary engine is employed. This can enable the concurrent evaluation of an entire population at once.

#### 6.2.4 Decision Maker

At the termination of a multi-objective search, the Pareto set will comprise a selection of non-dominated individuals, each satisfying the individual fitness components to a different degree. The user can either choose the ‘winning’ individual by hand, or a decision maker module can be defined.

Whilst keeping the user-based option available, an automated decision maker is defined as well. Although the three Minkowski functionals are optimised independently, visual similarity, and thus the overall quality of the individual, is a function of all three Minkowski functionals combined. As each fitness component (Minkowski error) is to be minimised, and each component can be deemed equally ‘important’, a Normalised Combined Fitness (NCF) can be defined to give an approximate measure of an individual’s overall quality. Each functional is normalised into the range  $[0,1]$  (using the minimum and maximum values of the relevant functional present in the population), and then summed together. This defines each individual’s NCF value. The individual with the minimum NCF is returned by the decision maker.

### 6.3 Robustness analysis

Before going on to apply this fitness function to a selection of target patterns, it is important to verify it in the same manner as the USM was verified for use in the CA-

based system. The Minkowski-based fitness function is analysed using the two-stage process described in chapter 4.

### 6.3.1 Fitness distance correlation

Similarly to the *Turbulence* system, each of the genes (input parameters) to the system have significantly different lower and upper bounds. A simple Euclidean distance would therefore introduce a false weighting in favour of the genes that naturally take larger values. For example, the range of the MR parameter is [5,50], meaning a distance of 1 is equal to 2% of its total range; the EL parameter, however, ranges [1,4], where a distance of 1 represents 33% of its total range. In order to provide a more meaningful distance value, therefore, we normalise each of the four distances to the range [0,1] and sum them together to form a single Normalised Combined Distance (equation 4.6). An arbitrary target is chosen from the dataset (specifically, that produced using parameters {35, 0.21, 1, 3} and shown in 6.5) and the 256-piece dataset described above analysed for Fitness Distance Correlation. The three plots (one for each objective) are shown in figures 6.6 - 6.8.

A clear correlation is evident in all three plots, though that for the area component (figure 6.6 is slightly confused by the row of points positioned at fitness 777643 (which represents completely covered, i.e. solvent saturated, behaviours). With these points removed, in fact, the correlation coefficient rises to 0.43, a value more in line with those of the other two fitness components which show much clearer correlations. These results give an encouraging indication that the Minkowski-based metric is an appropriate fitness function for use in this particular problem domain.

### 6.3.2 Clustering

Using the 256-piece data set described above, three square distance matrices were calculated, one for each fitness component. Although three different cluster trees could be produced, it would make the analysis of the tree very difficult, as visual similarity is a function of all three Minkowski functionals combined. As each component is to be minimised, and each component is equally important, these matrices can be

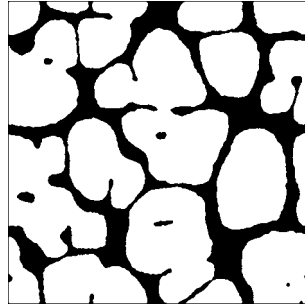


FIGURE 6.5: FDC ‘target’ individual (Parameters:  $\langle 35, 0.21, 1, 3 \rangle$ , Simulation time: 976s)

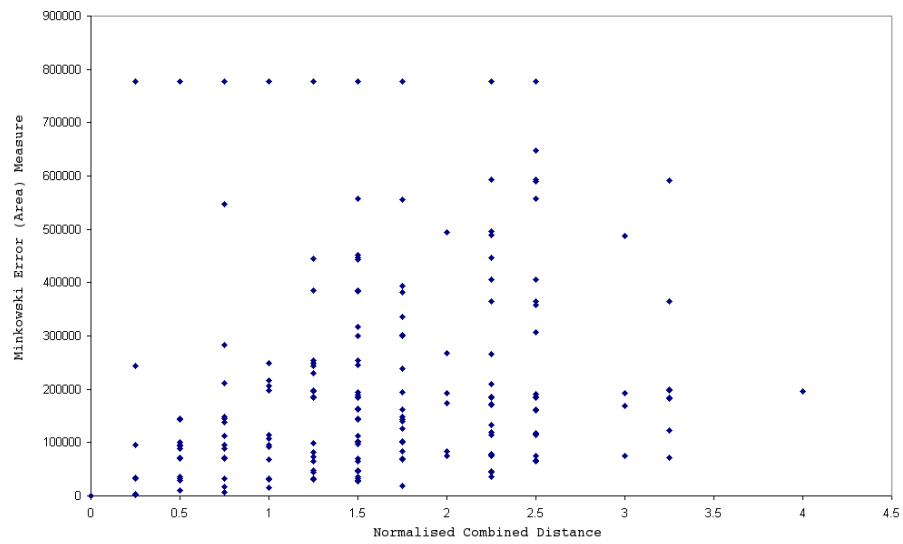


FIGURE 6.6: FDC scatter plot for the nano system (area only).  $r = -0.137$

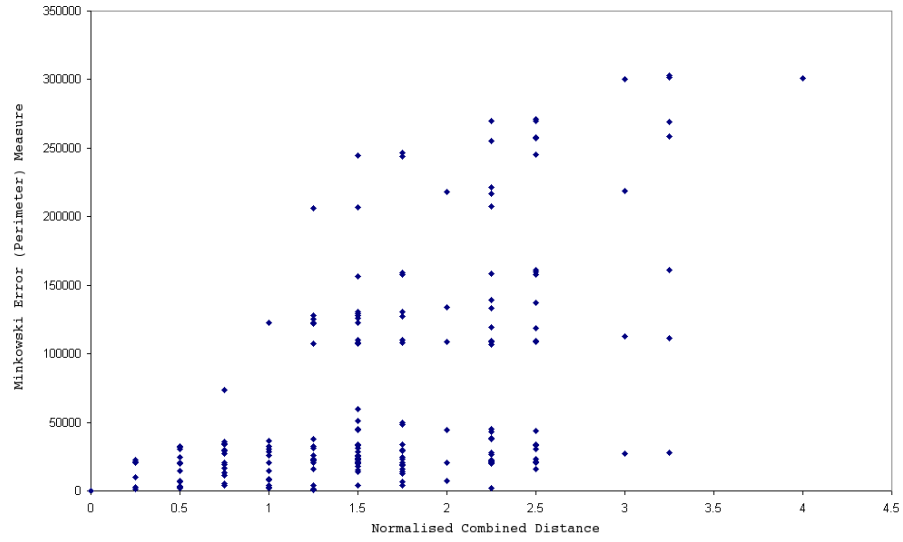


FIGURE 6.7: FDC scatter plot for the nano system (perimeter only).  $r = 0.548$

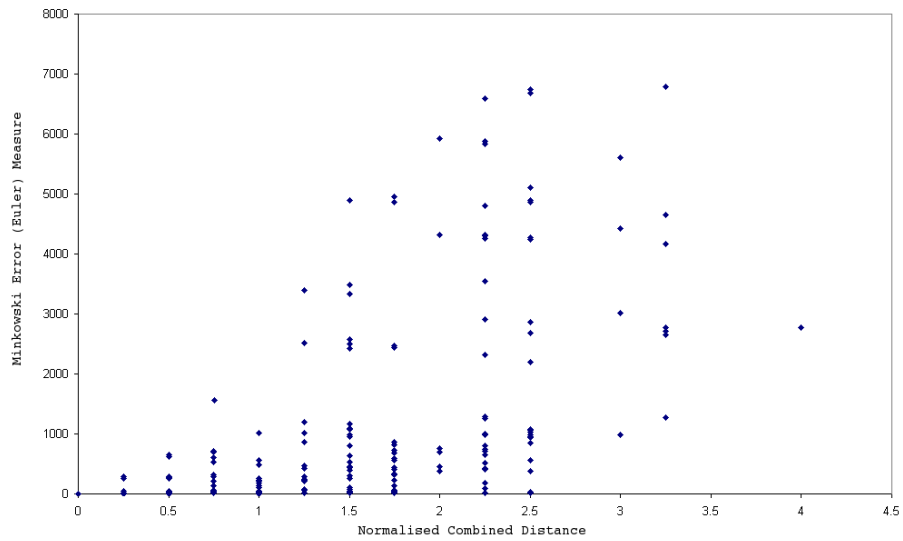


FIGURE 6.8: FDC scatter plot for the nano system (Euler only).  $r = 0.556$



combined into a single normalised combined matrix. Each of the three matrices is normalised so that every value is in the range  $[0,1]$ , and then summed together. It is this combined or ‘consensus’ matrix [5] that is used in the clustering algorithm. Figure 6.9 shows an annotated logarithmic tree representation of a Complete Linkage cluster. By analysing each leaf of the tree (which correspond to a single point in the dataset), it is clear that each of the seven main sub-clusters represents a particular morphological ‘family’ (shown by the annotations) and that similar families are close to each other in the tree. These results, along with those from the FDC analysis suggest that the Minkowski-based fitness methodology should be an effective method for evolving target behaviour in this particular complex system. We go on now, therefore, to show the results of using this method on four target patterns.

## 6.4 Results

With verification that the Minkowski-based fitness seems to be a robust method in this particular genotype–phenotype–fitness mapping, we move on now to test our evolutionary algorithm on a number of target patterns. The system was tested on a set of four contrasting target images, each taken directly from atomic force microscope images of real (i.e., non-simulated) nanostructures. The experiments presented below are therefore not only a test of the evolutionary algorithm and the Minkowski-based fitness, but also of the fidelity and representationl power of the simulator itself. Each target pattern represents a different ‘morphological family’. Both the original atomic force microscope images as well as the thresholded and despeckled versions used as targets in the evolutionary algorithm are shown in figure 6.10.

The MOGA was run, using the Minkowski-based fitness function on each target until the stopping criterion defined in section 6.2.3 was reached. Each experiment was run ten times. Figures 6.11 - 6.14 and table 6.2 show details of each MOGA run, as well the best result for each of the four targets, as selected by the automated decision maker described above.

In each case, the evolved patterns are very similar to the target morphologies represented by the AFM images. The connected nature of the “cell” and “labyrinth”

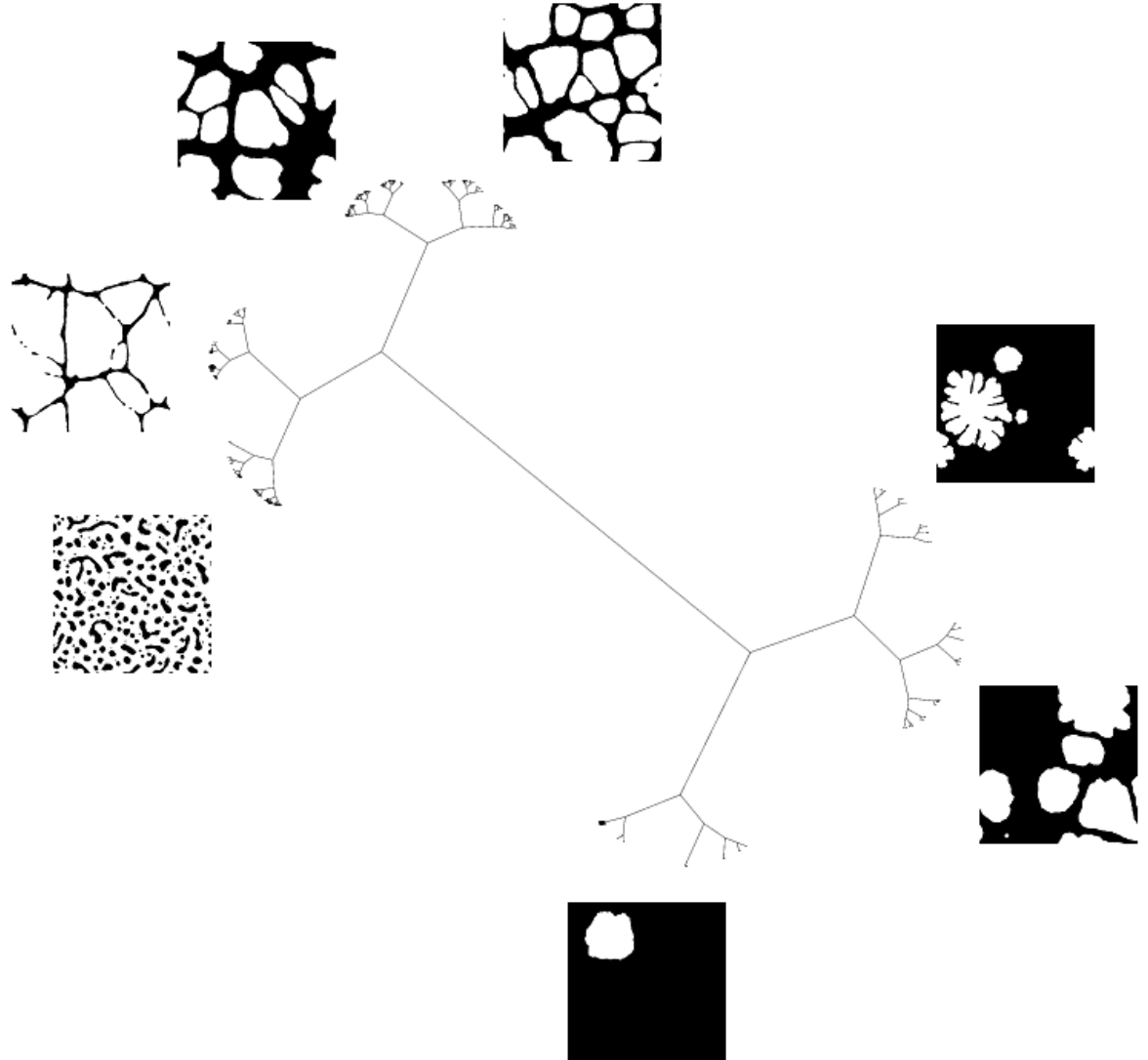


FIGURE 6.9: Logarithmic clustering tree showing the classification of the nano system dataset by our Minkowski-based similarity metric

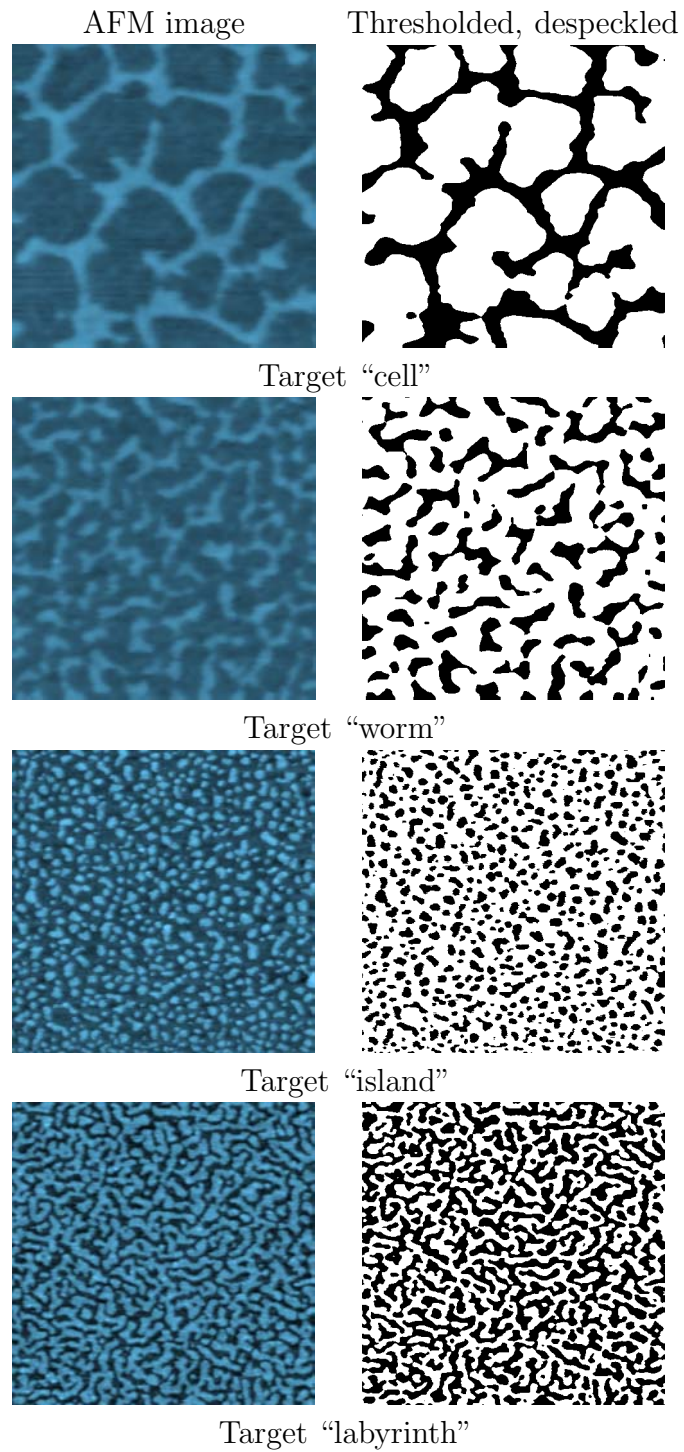


FIGURE 6.10: Nanostructured target patterns, both ‘real’ atomic force microscope images, and thresholded, despeckled, binary version for use within the evolutionary algorithm

Target	Run	Generations	NCF	Resultant genotype (ADM)			
Cell	1	38	0.00985	39.0	0.2	0.971	2.776
Cell	2	21	0.02122	40.0	0.2	1.094	3.218
Cell	3	52	0.00614	37.0	0.2	1.024	2.986
Cell	4	48	0.01267	43.0	0.2	0.704	2.034
Cell	5	54	0.00252	44.0	0.2	0.778	2.250
Cell	6	32	0.02562	40.0	0.2	0.834	2.389
Cell	7	15	0.01134	46.0	0.2	0.706	2.075
Cell	8	100	0.73927	7.0	0.3	2.000	3.581
Cell	9	14	0.78069	9.0	0.3	1.691	3.006
Cell	10	100	0.69911	10.0	0.3	1.080	1.836
	$\mu$	47	0.23084				
	$c_v$	0.659	1.52366				
Worm	1	43	0.00936	22.0	0.2	1.233	3.036
Worm	2	27	0.02964	25.0	0.2	0.899	2.171
Worm	3	31	0.05639	36.0	0.2	1.666	3.820
Worm	4	26	0.01675	25.0	0.2	0.990	2.312
Worm	5	26	0.03213	20.0	0.2	1.404	3.459
Worm	6	25	0.05003	35.0	0.2	1.464	3.538
Worm	7	52	0.94533	5.0	0.3	1.376	1.647
Worm	8	11	1.15517	14.0	0.3	1.667	1.864
Worm	9	16	1.10001	25.0	0.3	1.529	1.728
Worm	10	100	0.93187	5.0	0.3	1.748	2.099
	$\mu$	36	0.43267				
	$c_v$	0.714	1.20402				
Island	1	24	0.05998	14.0	0.2	1.361	1.575
Island	2	16	0.02623	11.0	0.2	1.076	1.255
Island	3	59	0.01330	7.0	0.2	1.396	1.673
Island	4	10	0.22058	6.0	0.2	1.464	1.837
Island	5	14	0.06140	6.0	0.2	1.959	2.407
Island	6	19	0.15581	5.0	0.2	1.807	2.241
Island	7	11	1.90158	9.0	0.3	1.491	1.456
Island	8	24	0.04980	13.0	0.2	1.612	1.755
Island	9	100	1.74470	5.0	0.3	1.790	1.808
Island	10	16	1.96128	8.0	0.3	1.643	1.622
	$\mu$	29	0.61947				
	$c_v$	0.97455	1.39832				
Labyrinth	1	10	0.22369	5.0	0.3	1.509	1.776
Labyrinth	2	62	0.02913	5.0	0.3	1.580	1.910
Labyrinth	3	10	0.14225	8.0	0.3	0.853	1.000
Labyrinth	4	100	0.04312	5.0	0.3	1.950	2.363
Labyrinth	5	23	0.16899	9.0	0.3	0.842	1.041
Labyrinth	6	45	0.09556	5.0	0.3	1.999	2.432
Labyrinth	7	20	1.69588	11.0	0.5	2.000	2.202
Labyrinth	8	10	0.12358	5.0	0.4	1.610	1.780
Labyrinth	9	17	1.47631	5.0	0.5	1.300	1.000
Labyrinth	10	12	1.47771	6.0	0.5	1.389	1.057
	$\mu$	31	0.54762				
	$c_v$	0.964	1.27189				

TABLE 6.2: Statistical analysis of Nano system evolutions

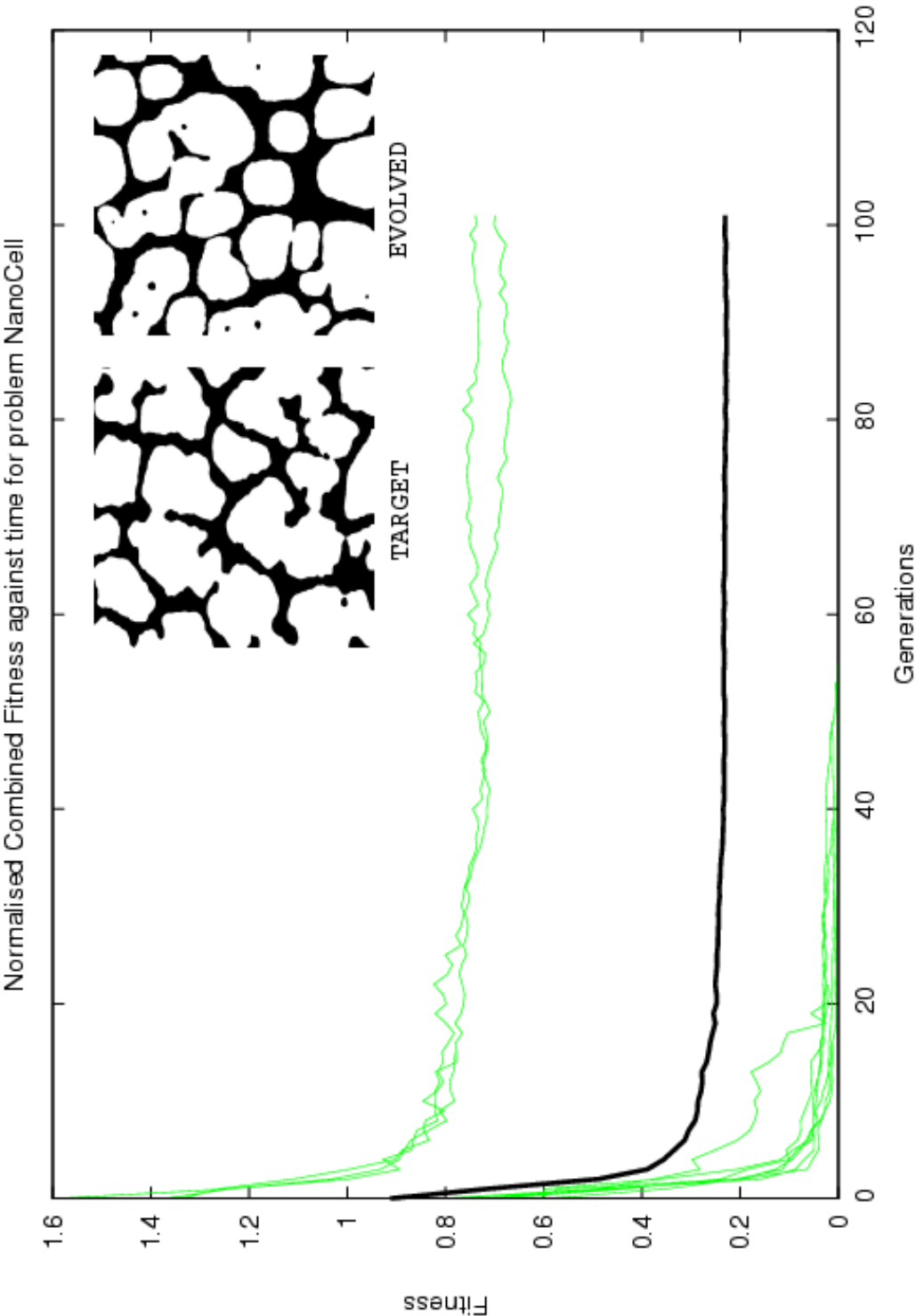


FIGURE 6.11: Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “cell” target

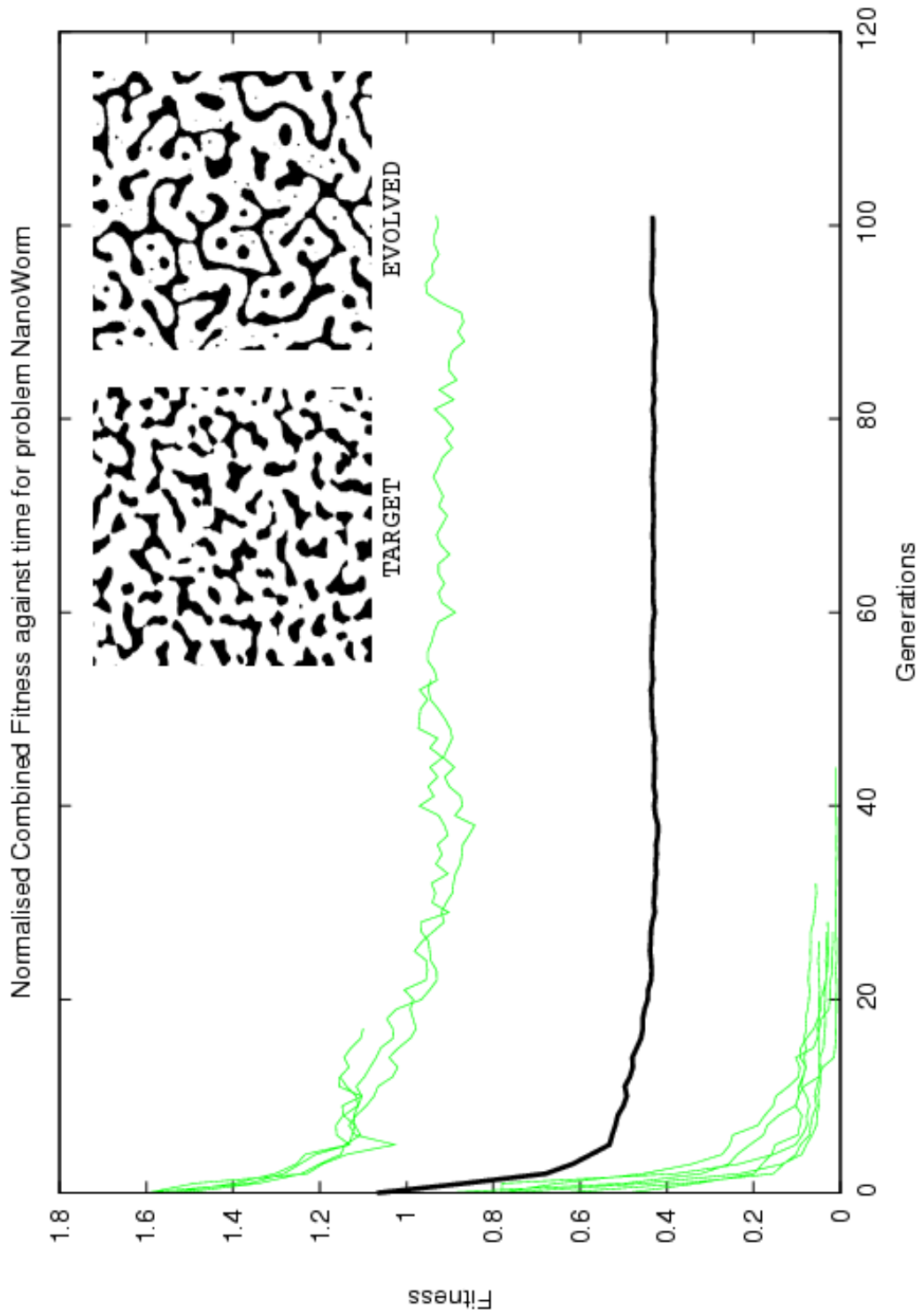


FIGURE 6.12: Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “worm” target

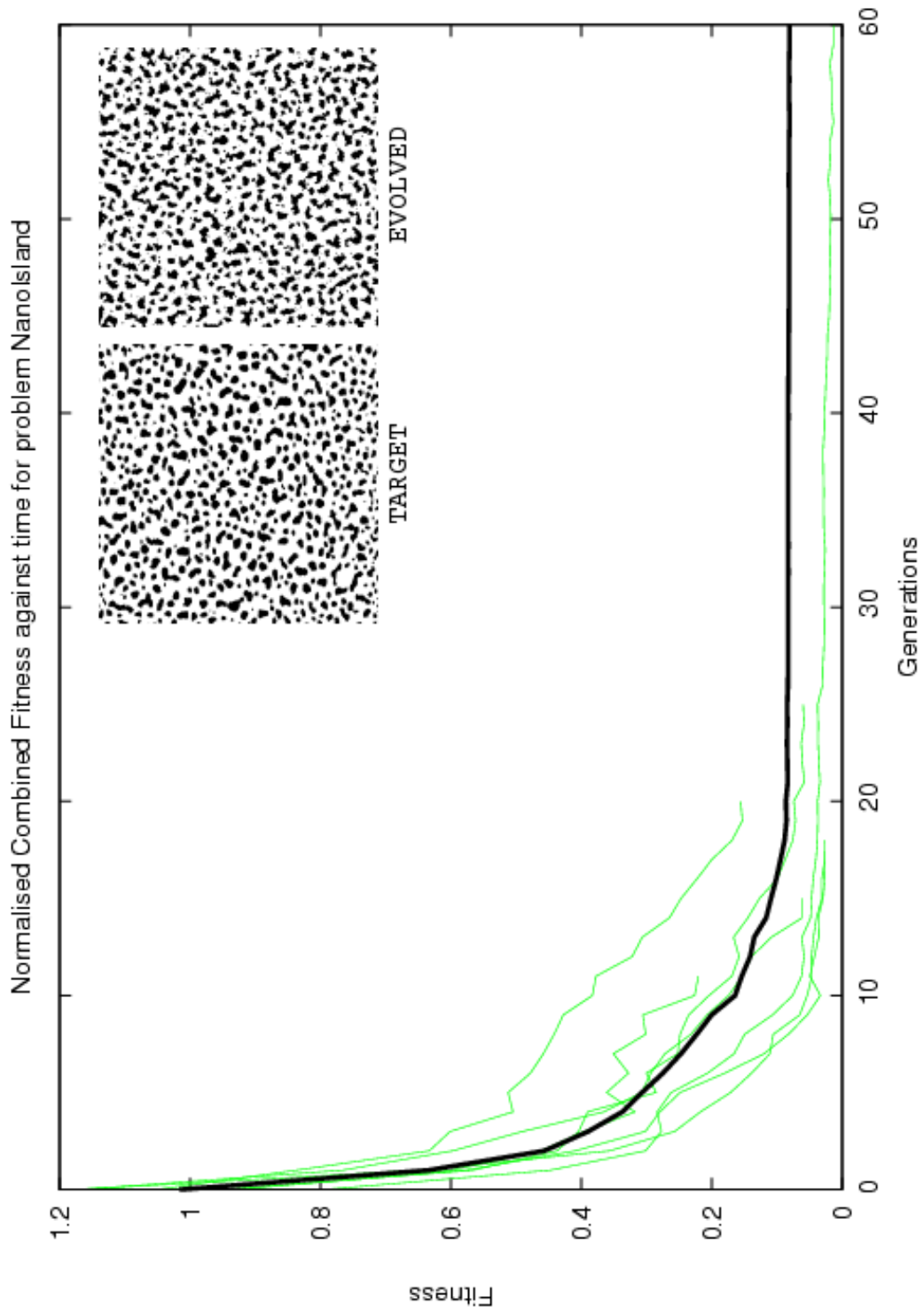


FIGURE 6.13: Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “island” target

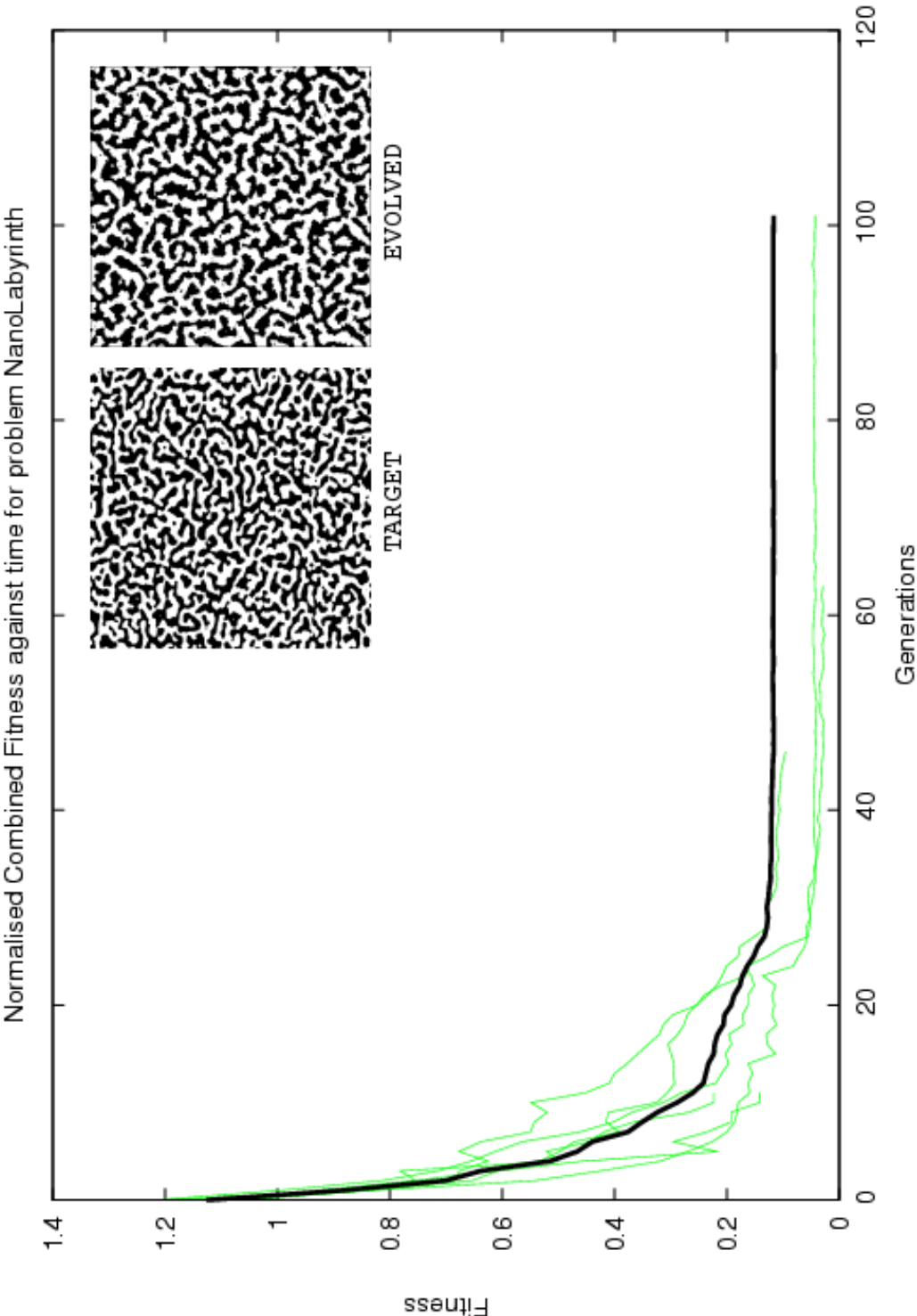


FIGURE 6.14: Target pattern, evolution graph and best resultant pattern from the evolutionary algorithm, running on the “labyrinth” target



patterns are well-matched, as is the rather more disconnected trend of the “worm” and “island” patterns. The small length-scale of the “island” and “labyrinth” patterns are well contrasted with the larger scale of the “cell” and “worm” behaviours.

In all four experiments, there is a reasonably high degree of variation between the number of generations taken to reach the stopping criteria. This is not altogether surprising, however, as searches that, by virtue of their initial population, start closer to an optimum area of the search space than others, will naturally take fewer generations to converge.

There is also considerable variation in the NCF values among the ten runs for each experiment. This suggests that the search space itself is highly uneven, with local optima or very different depth and with a number of optima for a given target – some optima will be of higher quality than others. The results for the “cell” and “worm” targets both show two distinct shapes of trajectory, with most of the ten runs following the higher quality (i.e., lower resultant similarity value) trend, but with three or four runs following a path that converges on a lower quality (but still visually acceptable) result. This may indicate that there are two distinct areas of the search space that correspond to patterns of this type, with one satisfying the Minkowski similarity measure to a greater extent than the other. It should be noted that our population size is relatively small (20 individuals), due to the constraints of the parallel computation facility – a larger population, it is expected, would ensure that each search would visit at least one of the higher quality optima. As in the initial experiments with the cellular automata, there is clear evidence that the search space is multi-modal, i.e., that two different genotypes may give very similar phenotypes (and fitnesses).

## 6.5 Identifying limitations in Minkowski image analysis: Multi-phased nanostructures

In this section, preliminary work is shown where the methods employed above are applied to more complex class of target pattern - a two-phased simulation where a ‘foreground’ and ‘background’ pattern are formed, each with their own characteristics.

Name	Type	Lower	Upper
MR	integer	5	50
coverage	real	0.05	auto
kT	real	0.5	2
EL	real	1	4
mu	real	-2.5	-2
A	integer	50	1000
B	real	0	1
vc	real	0	1

TABLE 6.3: Extended nanostructure simulator: Parameters

A method of phase separation and an incremental evolutionary process is suggested for the evolution of such behaviours as the basis for further investigation.

As can be seen from figure 6.3, some of the patterns formed by this physical system can be seen to be in two distinct phases such as, for example, a smaller scale collection of disconnected islands *within* a connected, cellular superstructure. The authors of the Monte Carlo model extended it to represent this type of self-organisation process by the addition of a dynamic chemical potential parameter. This extended model is driven by eight parameters. Details of all eight parameters are given in table 6.3. The first four are identical to those in the single-phased system described above; the latter four are related to the specification of the dynamic chemical potential variable, with  $\mu$  governing the initial potential level.  $A$  defines the gradient between the two phases, i.e., whether the switch between phases is sudden or gradual, and  $B$  defines the nature of the final chemical potential value once the switch has completed.  $vc$  defines the position of the switch in time.

### 6.5.1 Methodology

Simply ‘plugging in’ the extended simulator to the Minkowski-based objective function described above would not be appropriate. The Minkowski functionals are an average measure across a given image – the area, perimeter and Euler characteristics of the whole image. A two-phase image, where each phase has a significantly differ-

ent Minkowski signature, clearly poses a considerable problem. A closer investigation of the parameter set provides the route to a potential solution. Specifically, the  $vc$  parameter can be interpreted as a measure of the size ratio between the two phases. Setting this parameter to its lower bound value of 0 results in the production of the foreground pattern only, whilst the upper bound value of 1 results in only the background pattern. A value between these two extreme bounds results in the two-phased pattern such that  $vc$  equals the size ratio of the foreground pattern to the total image size. A target pattern,  $T$ , can therefore be partitioned into two single-phased images. If a parameter set can be found that produces both separated patterns when  $vc$  is set to 0 and 1 respectively, the target pattern can be produced by analytically setting the  $vc$  parameter as described above.

The target pattern is manually separated into its two phases at the outset of the search, by hand. An example of this separation process is shown in figure 6.15. Note that in the background-only pattern, the areas formerly occupied by the foreground pattern have been manually extrapolated from the surrounding background pattern.

The methodology then continues using an ‘incremental complexity’ method: Two instances of the GA are run – one concerned with matching only the foreground pattern and one concerned with matching only the background pattern. That is to say, all parameters are evolved, with the exception of the  $vc$  parameter which, in the former case is set always to 0, and in the latter set always to 1. These two instances are run until a stopping criterion identical to that defined in 6.2.3 is reached. The final populations of these two runs are then combined, and re-evaluated, but considering *both* foreground and background comparisons. In this way, a highly ranked behaviour would be one that reproduces both foreground *and* background patterns faithfully. Setting the  $vc$  parameter such that the relative sizes of the two phases are well-matched (simply by measuring the size ratios in the two-phased target pattern), should result in a good match for the two-phased target.

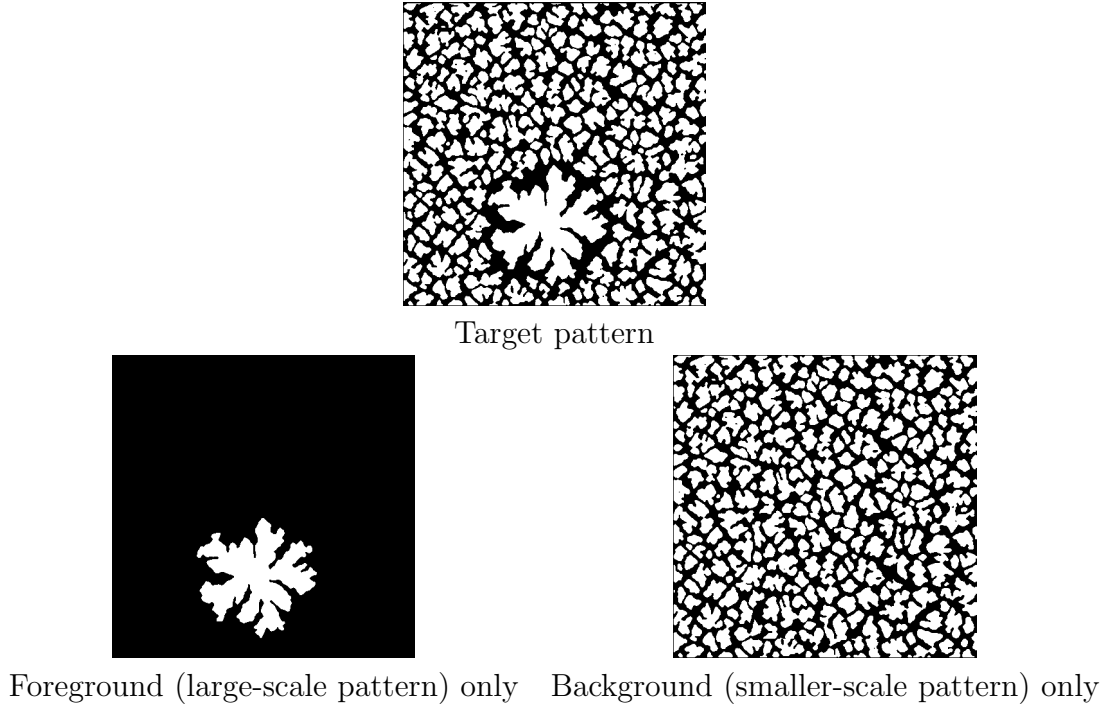


FIGURE 6.15: Extended nanostructure evolution: Phase separation

### 6.5.2 Results

Figure 6.15 shows an example two-phased nanostructured pattern. The MOGA was run in the iterative manner described above, using the phase-separated Minkowski-based fitness function on this target until the stopping criteria defined in section 6.2.3 was reached.

Figure 6.16 shows the final result, as selected by the automated decision maker.

### 6.5.3 Outcomes of the multi-phase investigations

It is perhaps unsurprising that the results for this two-phased target are not as impressive as those for the single-phased system studied previously. It is a much more complex search space – a set of parameters must now satisfy two conditions simultaneously, and the multi-modal nature of the space mean that a given set of parameters may reproduce the background pattern very faithfully indeed, but not get close to the foreground pattern, or vice versa. With this in mind, the result shown in figure 6.16

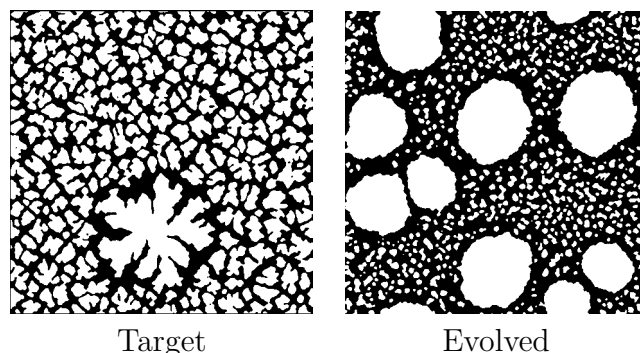


FIGURE 6.16: Evolved two-phased nanostructure

could actually be seen to be surprisingly good for a relatively simple metaheuristic (even with a complex fitness function) – the foreground pattern is lacking the fingered-style edging, and there are too many areas of white space, but the general morphology is reasonably close to what is required; the background pattern is a cellular structure, as required, though on a rather smaller-scale than the target. Considered as a whole, although the evolved pattern is not as precisely reproduced as the single-phased results presented earlier, we can surely say that the general morphology is in the right ‘ball park’. This may well be an example of a search space where a more sophisticated metaheuristic might be more effective than the standard GA currently implemented in the Evolutionary Engine, and this is certainly grounds for further research.

## 6.6 Conclusions

The results presented in this chapter show that Minkowski functional analysis in the context of a MOGA seems to be a highly successful methodology in the evolution of simulated nanostructure patterns. Although the mapping from simulator parameters onto those of the actual laboratory equipment is not a particularly strong one, these results would seem to suggest that such a methodology could be well suited to the optimisation of real physio-chemical systems, taking us one step further in the development of software control of matter. If one draws an analogy between visual similarity and, for example, electrical behaviour/function, the ‘real world’, “dial-a-

function” applications are clear. In any such application, however, the time and computational expense of the algorithm is crucial, and as it stands, run-time may be considered to be prohibitively lengthy. Run-times for each experiment are given in table 7.10, showing run-times of up to 100 hours in some cases. In the context of the real-time chemical optimisation we hope to consider in the future, this time may also equate to financially expensive laboratory and equipment time. We move on now, therefore, to consider a method by which the speed of the evolutionary process could be increased, but without significant compromise to the quality of the results.

## CHAPTER 7

# Accelerated evolution through fitness approximation

Notwithstanding the quality of the results presented above, the time taken, both by the simulator itself, but also by the Minkowski analysis, is not inconsiderable. This chapter explores the potential for savings in time and computational expense through the use of a fitness approximation model. It is shown that a neural network ensemble can accurately embody not only the behaviour of the complex system, but also its subsequent mapping onto the Minkowski analysis methods.

A forty generation run of the genetic algorithm presented in chapter 6, even when parallelising the evaluation of a single population, takes in the region of sixteen hours, of which the actual genetic activity is a minute proportion. Such a ‘bottle neck’ is a common problem in evolutionary computation, especially that related to the optimisation of complex systems such as the simulator with which we are dealing. In a number of cases, approximations of the fitness function can be very useful. The author of [61] presents an excellent survey of this field of fitness approximation.

One particularly pressing concern in the application of approximation models to evolutionary computation is that the landscape topology of the model and that of the real system should be as congruent as possible. If not, a search of the model may converge on an optimum that is not represented by the real system, and therefore entirely redundant. Constructing a functionally faithful model is therefore of prime importance.

As described in [61], there are a number of techniques available for approximating a fitness function, including polynomial and kriging models, neural networks and support vector machines. Neural networks are a particularly appealing modelling technique as they are nature-inspired, and thus sit very comfortably alongside the evolutionary computation methods employed in this methodology. Moreover, the underlying architecture and function of an artificial neuron is relatively simple, their computational power being produced by the aggregation of these simple components into a complex network.

## 7.1 Artificial neural networks

Artificial neural networks [36] have shown to be particularly effective tools for function approximation [59], and indeed have also been employed as fitness surrogates [62, 7, 30]. Artificial neural networks are a nature-inspired method of computation based closely on the architecture of a biological neuron in which, as illustrated in figure 7.1, an electrical signal arrives through an axon at a synapse; if the signal is of sufficient magnitude, it will stimulate synaptic chemicals to move across the synapse, in effect, passing the signal through a dendrite into the body (soma) of the neuron, where some processing can occur which may result in a further signal being passed through one of the output axons.

Artificial neurons, first proposed in 1943 by McCulloch and Pitts [82], simulate these basic functions of natural neurons. A collection of input signals arrive through the synapses of the artificial neuron. Different input synapses can be given different weightings. The neuron then performs some function on these input signals (the so-called ‘activation function’). It is the output of this function that is passed on to any adjoining neurons. Whereas traditional computation methods merely implement a sequence of logical and arithmetic operations, artificial neural networks have the ability to learn from examples and adapt their behaviour. This process is described in more detail below.

McCulloch and Pitts’s original proposal represented only binary input/outputs. That is, the neuron either fires (output of one) or does not (output of zero). Neu-



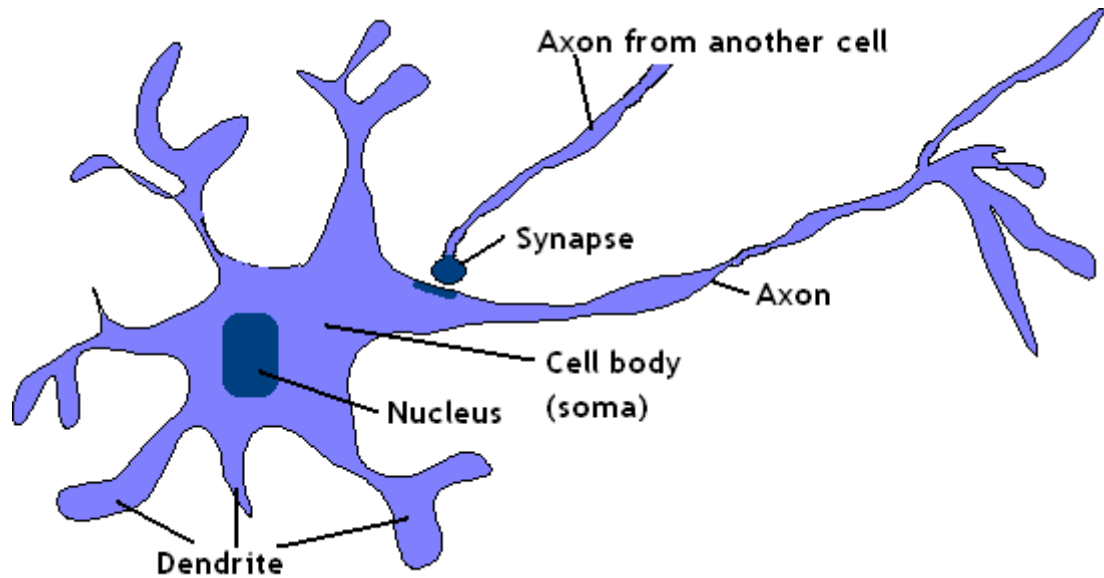


FIGURE 7.1: Schematic of a biological neuron

rons in a McCulloch-Pitts network are connected by directed, weighted paths that is considered excitatory if the weight on the path is positive; otherwise it is inhibitory. The activation function mentioned above can be as simple as a step function, that is, if the total input to the neuron is greater than some threshold, an output signal is generated. This simplest of neurons can be used to represent basic logic operations, as shown in figure 7.2, where the step function threshold in both cases is 1.

The McCulloch-Pitts Neuron forms the foundation of modern artificial neural networks, but a number of changes have been made to allow the learning process mentioned above. Modern networks are usually arranged in layers. The simple examples in figure 7.2 are single layer networks, but these are significantly restricted in the complexity of function they can represent. Multilayer networks add hidden layers in addition to those for input and output, to allow the learning of nonlinear relationships – a vital criterion for our purposes. It has been shown that a network with hidden layers can approximate any continuous function [112]. Indeed, as explained in [60], such networks can be termed ‘universal approximators’.

The success of an artificial neural network in modelling a continuous function is

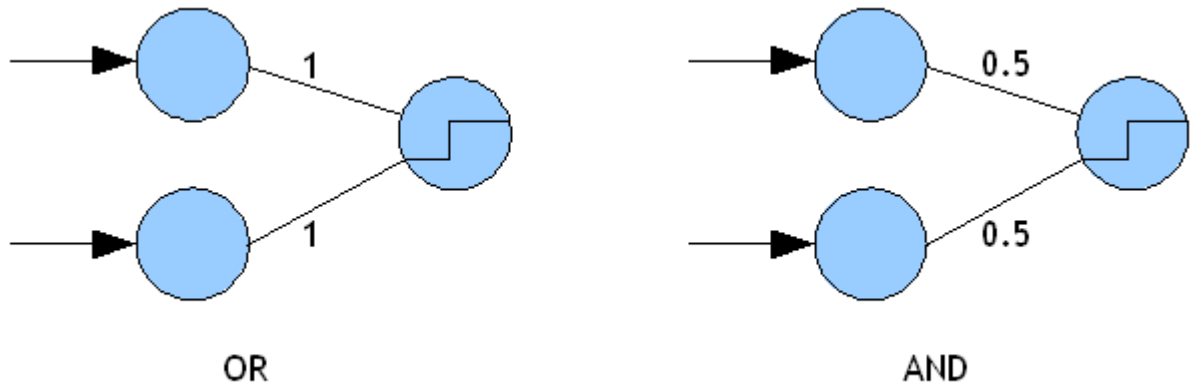


FIGURE 7.2: Neural network implementations of logic gates. Assuming a step threshold ( $> 1$ ) in both cases, it is evident that the OR example functions as an OR logic operator – if either input fires, the output value will be 1, if both fire, it will be 2. Similarly, in the AND example, both inputs must fire if the output sum is to reach the threshold of 1.

highly dependent on the number of hidden layers, and indeed the number of nodes in each layer. As mentioned already, a network with a single hidden layer can approximate any continuous function, but two or more can be beneficial to certain problems. Too many hidden nodes can lead to overfitting (i.e., the networks perform well within the training sample, but poorly out-of-sample), too few makes it difficult to learn any patterns in the data. The simple threshold function used in the simple examples above is just one possible activation function that could be used in an artificial neuron, as discussed below.

### 7.1.1 Learning

Learning in artificial neural networks is achieved by modifying the synaptic weights in the network, which are initially randomly selected. Given a training set of input data and target output values, the network under consideration processes the input values and compares the resulting output with the desired values. Depending on the size of the error, the weights are adjusted using an appropriate learning algorithm. This process is repeated until the error is considered to be acceptably small. Although

in some cases, the acceptable error may be no more than zero, in the majority of problems, this is not feasible, in particular as care must be taken to avoid the problem of overfitting the data, as mentioned above.

A multi-layer neural network trained using the Backpropagation learning algorithm [51] is one of the most powerful forms of supervised neural network system. It is important to note that the choice of activation function to use in a backpropagation network is limited to functions that are continuous, differentiable and monotonically non-decreasing. Furthermore, for computational efficiency, it is desirable that its derivative is easy to compute. Usually the function is also expected to saturate, i.e. approach finite maximum and minimum values asymptotically. One of the most typical activation functions used, therefore, is the binary sigmoidal function:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (7.1)$$

where the derivative is given by:

$$f'(x) = (f(x))(1 - f(x)) \quad (7.2)$$

During the feedforward phase of the learning algorithm, each of the input units ( $X_i$ ) is set to its given input pattern value. Each input unit is then multiplied by the weight of its connection. The weighted inputs are then fed into the hidden units ( $Y_0$  to  $Y_j$ ). Each hidden unit then sums the incoming signals and applies an activation multiplied by the weight of its connection:

$$Y_j = f(\sum X_i W_{ij}) \quad (7.3)$$

The weighted signals are fed into the output units ( $Z_0$  to  $Z_k$ ). Each output unit then sums the incoming signals from the hidden units and applies an activation function to form the response of the net for a given input pattern.

During training, each output unit then compares its output ( $Z_k$ ) with the required target value ( $d_k$ ) to determine the associated error for that pattern. Based on this error, a factor  $\delta_k$  is used to distribute the error at  $Z_k$  back to all units in the previous layer:

$$\delta_k = (f'(Z_k))(d_k - Z_k) \quad (7.4)$$

Each hidden unit then computes a similar factor  $\delta_j$  that is a weighted sum of all the backpropagated delta terms from units in the previous layer multiplied by the derivative of the activation function for that unit:

$$\delta_j = (f'(Y_j))(\sum \delta_k) \quad (7.5)$$

After all the delta terms have been calculated, each hidden and output layer unit updates its connection weights and bias weights accordingly (where  $\Delta W_{ij}$  is the change in weight on the connection  $ij$ )

$$\Delta W_{ij}(t+1) = \eta \delta_j X_i + \alpha \Delta W_{ij}(t) \quad (7.6)$$

where  $\eta$  is a learning rate coefficient that is given a value between 0 and 1 at the start of training. The final term of the update function,  $\alpha \Delta W_{ij}(t)$  is an optional enhancement that adds in a so-called ‘momentum’ term,  $\alpha$ , that involves the previous weight change as a parameter. This momentum term means that shallower regions of the error surface are explored comparatively faster (with a smaller number of large steps), while the step size is decreased in steeper regions. This innovation can significantly increase learning speed. In addition to defining an appropriate network architecture, then, the proper selection of these two learning parameters,  $\eta$  and  $\alpha$  is crucial in finding the true global minimum error. After each epoch of training (that is, one presentation of the entire training set) the performance of the network is measured by computing the root mean square (RMS) error of the network for all of the patterns in the training set.

The question of when to halt training is another important issue. Very often, on completion of an epoch, the RMS error will also be computed for a set of data not included in the training set – a ‘validation’ set. Training can be ended when the error value for the validation set begins to rise. This is a simple yet effective method of preventing the network from being overtrained (i.e. simply ‘memorising’ the dataset rather than inferring generalisable patterns).

## 7.2 Artificial neural networks as fitness approximators

Three standard, general, multi-layer networks are constructed (using the *Joone* framework [79]) - one for each component (Minkowski functional) of our fitness function. The networks are trained using the backpropagation method. The inputs into the network are identical to the input parameters of the simulator, and the single output of each network is the relevant Minkowski functional. In this way, not only does our model stand in place of the simulation process, but also of the Minkowski analysis. The three networks in ensemble are then used in the parameter optimisation MOGA as in the previous chapter.

For maximum generalisability, it is important that the neural network ensemble can predict accurate output values across the search space. The dataset used for training and validation must therefore be representative of the entire parameter space. To this end, the same dataset as that used for the robustness checking described in section 6.2.1 was employed.

Initial training experiments showed that, though the area and perimeter functions could be reasonably well-learned by a neural network, the Euler function was considerably harder, due to the fact that the range (as a function of the mean) of output values is much larger than the other two functionals. To ease the learning process of the Euler characteristic, therefore, the function was manually ‘smoothed out’ by a) converting all values to be positive (although this would mean negative images would return the same Euler characteristic, when used with the other two networks, the area functional in particular would be able to differentiate such pairs of images) and b) cutting all values larger than 1000 to 1000.

### 7.2.1 Network parameters

As discussed above, the performance of a neural network is entirely dependent on a suitable architecture and set of learning parameters. To identify the architecture and parameters that produce the most faithful networks, we coupled the neural network system to the Evolutionary Engine (as proposed in [17]), where the chromosome is of the form shown in table 7.1.

Name	Type	Lower bound	Upper bound
Learning rate	float	0	1
Momentum	float	0	1
Nodes in hidden layer 1	int	0	15
Nodes in hidden layer 2	int	0	15
Nodes in hidden layer 3	int	0	15
Nodes in hidden layer 4	int	0	15
Nodes in hidden layer 5	int	0	15
Nodes in hidden layer 6	int	0	15

TABLE 7.1: Chromosome for the evolution of neural network parameters

Using the Evolutionary Engine, it is possible to evolve, therefore, the optimal values for the learning rate and the momentum term for the backpropagation learning process, as well as the number of hidden layers (to a maximum of 6) and the number of nodes they contain. The fitness for each parameter set/architecture is calculated as the mean overall RMS error of the network over five evaluations (to compensate for the stochasticity introduced by the randomly assigned initial weights of the network).

The dataset is partitioned into two, a training set (75% of the samples) and a validation set (25% of the samples). With each of the five evaluations, this partitioning is randomly redefined to further avoid overfitting to a particular subset of the data. The training process is set to stop when the validation error starts to rise for five consecutive iterations. Fitness is defined simply as the root mean square training error of the network. Note that even when distributed over the compute cluster, the slow speed of the backpropagation learning process means that, in the interests of computation time, the number of generations is kept low – the stopping criterion is a simple cap at twenty generations.

A GA was run for each of the three networks (one for each of the three Minkowski functionals), and each was run ten times. Graphs of the average fitness over time are shown in figures 7.3 - 7.5, with a table of statistics at table 7.2.

The best (lowest global RMSE) of each of the ten runs was selected for each of the three evolved networks.

The general fidelity of each network can be estimated by taking the average RMS

Network	Run	RMSE	Resultant genotype		
			Learning	Momentum	Number of nodes
A	1	0.02722	0.07310	0.16998	13-5-3-8-10
A	2	0.02428	0.04167	0.19006	11-10-7-9
A	3	0.02305	0.05301	0.11543	12-11-10-10
A	4	0.02478	0.08234	0.0	15-13-12-8-1
A	5	0.02419	0.07565	0.02154	10-6-9
A	6	0.02644	0.05649	0.10981	14-4-8-6-13
A	7	0.02192	0.02334	0.26028	12-11-12-4
A	8	0.02329	0.04511	0.13735	14-12-11-13
A	9	0.02940	0.02362	0.60684	14-8-3-14-7
A	10	0.04644	0.08934	0.00335	14-3-14-9-10
	$\mu$	0.02710			
	$c_v$	0.00714			
P	1	0.01715	0.22062	0.01721	13-8-13-6-15
P	2	0.01873	0.04766	0.76967	14-14-13-15-14
P	3	0.01542	0.12598	0.36757	15-11-3
P	4	0.01688	0.14792	0.13110	12-5-14-10
P	5	0.01850	0.11366	0.13645	14-6-8-3-7
P	6	0.01625	0.09911	0.50339	13-10-15-12-7
P	7	0.01739	0.07694	0.44522	12-11-6-10-5
P	8	0.01626	0.14608	0.21958	14-5-11-2
P	9	0.01753	0.12831	0.30158	12-6-7-9-8
P	10	0.01750	0.09029	0.09611	10-7-15
	$\mu$	0.01716			
	$c_v$	0.00102			
E	1	0.10768	0.05684	0.06380	12-13-12
E	2	0.08933	0.01164	0.32716	11-14-6-9
E	3	0.10173	0.023	0.00684	14-5-11-8-11
E	4	0.09284	0.02394	0.16329	13-11-5-14-14
E	5	0.10959	0.07700	0.01517	11-12-5
E	6	0.10627	0.04553	0.11062	13-10-14-10-13
E	7	0.11786	0.05444	0.06527	9-8-12-12-7
E	8	0.18965	0.21762	0.05681	13-4-13-6-14-13
E	9	0.11879	0.01625	0.31573	7-12-7-4-9
E	10	0.36174	0.34062	0.30308	10-14-6-15-9
	$\mu$	0.13955			
	$c_v$	0.08296			

TABLE 7.2: Statistical analysis of neural network evolutions, showing the resulting parameters as well as the RMS training error

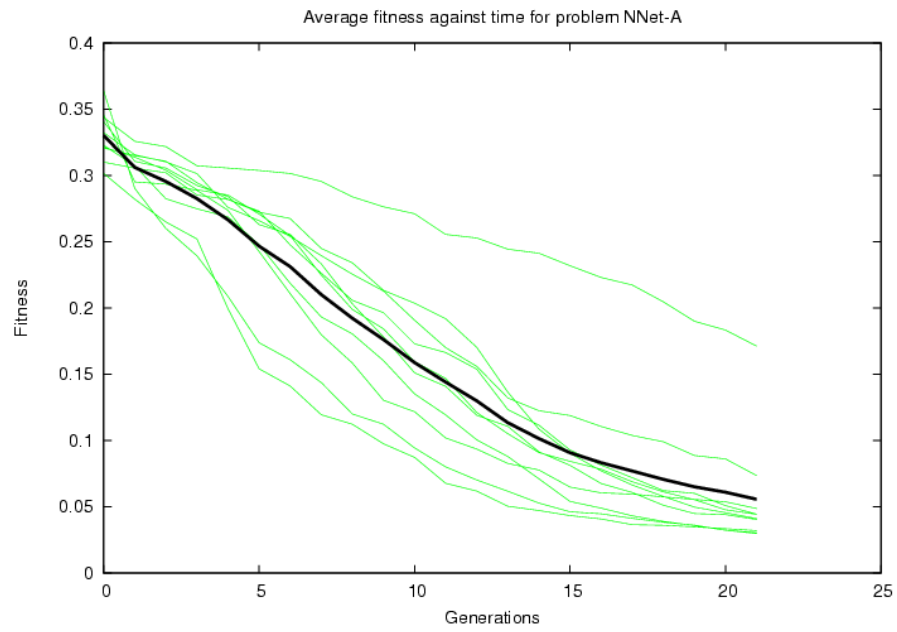


FIGURE 7.3: Graph of fitness against time for the evolution of the neural network approximating the Minkowski area of a nano simulator pattern

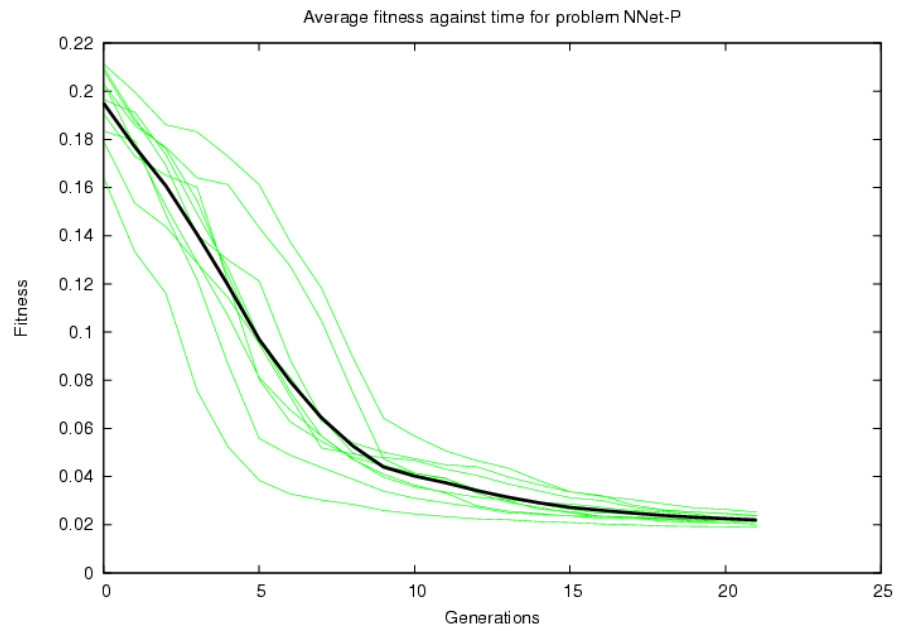


FIGURE 7.4: Graph of fitness against time for the evolution of the neural network approximating the Minkowski perimeter of a nano simulator pattern



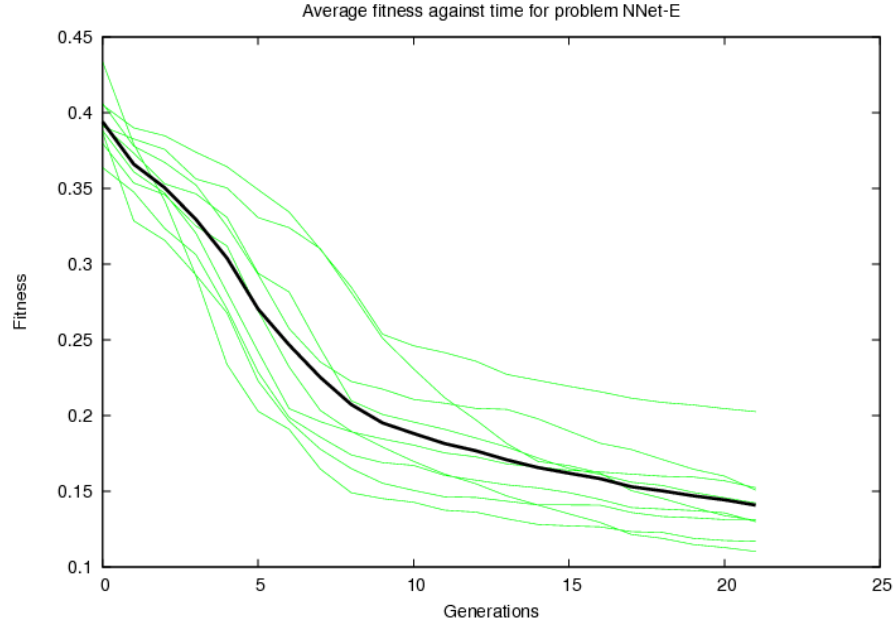


FIGURE 7.5: Graph of fitness against time for the evolution of the neural network approximating the Minkowski Euler characteristic of a nano simulator pattern

error across the entire dataset. The Area network is the most accurate, at a very impressive 96.5%. The Perimeter network also shows an excellent degree of fidelity to the original function, with an accuracy of 87.0%. As expected, the Euler network is not as accurate, with a fidelity value of only 46.6%. Considered as a group, however (that is, always being used together to calculate all three functionals), the overall mean accuracy is an encouraging 76.7%. In the next section, we show the results of evolutionary runs using the three neural network approximators in place of the simulator and Minkowski analysis.

### 7.3 Results

The experiments presented in section 6.4 were re-run on the same four targets, keeping all parameters identical, but using our newly evolved group of neural network approximators (one for each Minkowski functional) in place of the Monte Carlo simulator and Minkowski analysis module.

As before, each experiment was run ten times. Table 7.3 shows statistical details

of each MOGA run, whilst figures 7.6 - 7.9 shows the graph of evolution and the best result for each of the four targets, as selected by the automated decision maker described in 6.2.4.

In each case, the evolved patterns are very similar to the target morphologies represented by the AFM images. The connected nature of the “cell” and “labyrinth” patterns are well-matched, as is the rather more disconnected trend of the “worm” and “island” patterns. The small length-scale of the “island” and “labyrinth” patterns are well contrasted with the larger scale of the “cell” and “worm” behaviours. This is particularly impressive when one is reminded that the model is standing in place of both the simulator and the Minkowski analysis. It suggests that such an approximation technique could be very useful as part of an ‘evolutionary chemistry’ system.

As in the experiments using the ‘real’ system, there is a reasonably high degree of variation between the number of generations taken to reach the stopping criteria, but a lesser degree of variation amongst the NCF values for each run. This suggests that the ‘approximated’ search space may be rather less uneven than the ‘real’ landscape, although the quality of results shows that they are still congruent enough to be very effective. However, the highly jagged trajectory of one run for the “cell” target in particular (as shown in figure 7.6) shows that the approximated space may be harder to search in certain areas.

Both the statistical analysis table and the resultant target patterns bear a striking resemblance to those for the experiments presented in chapter 6; the time taken to for each GA run, however, was decreased from the region of several hours to a few seconds. This is summarised in table 7.10.

## 7.4 Conclusions

This chapter has shown that a neural network ensemble can accurately embody not only the behaviour of the complex system, but also its subsequent mapping onto the Minkowski analysis methods. This innovative approximation method results in a dramatic saving of time with little compromise to the quality of results.

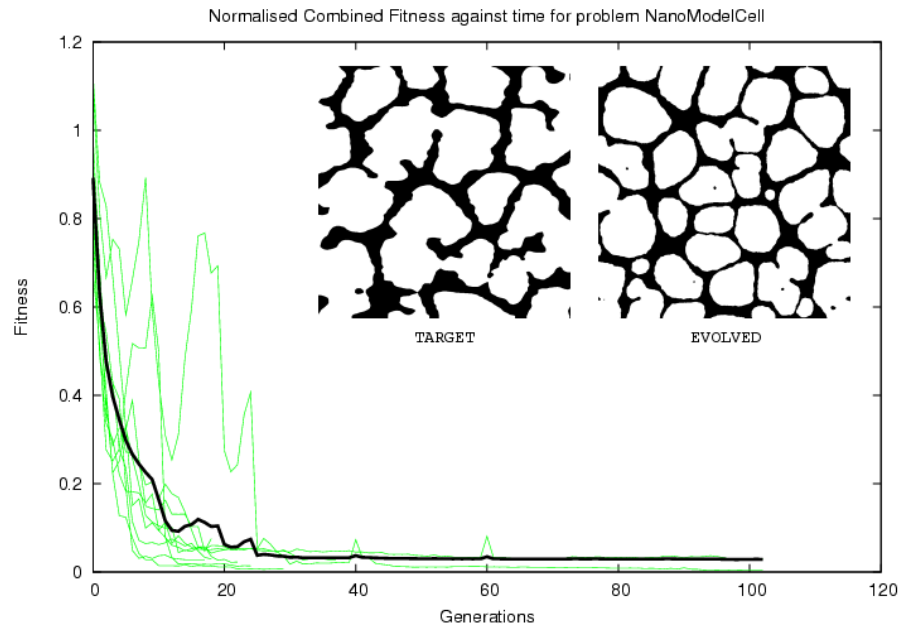


FIGURE 7.6: Evolution of nanostructures with surrogate fitness - target “cell”

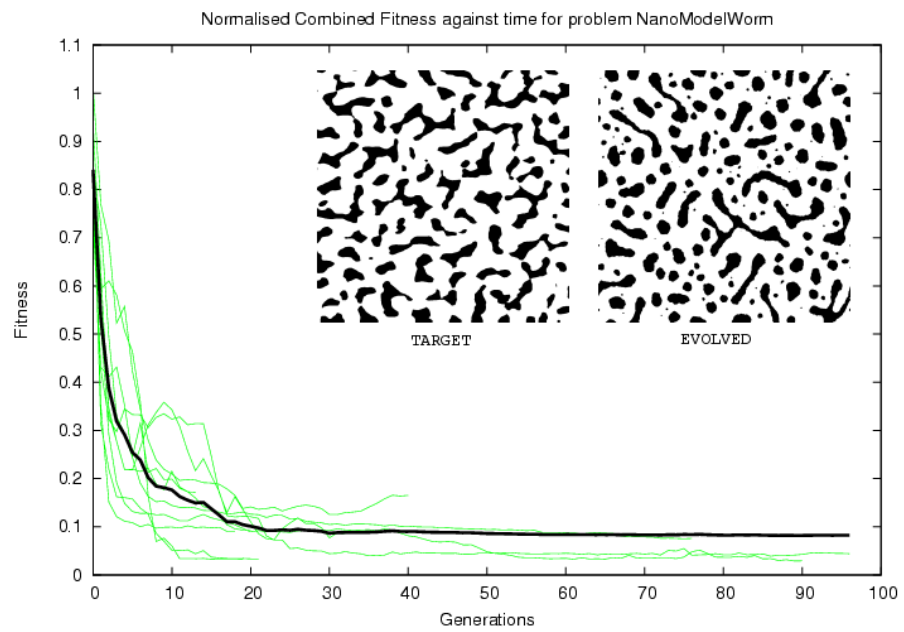


FIGURE 7.7: Evolution of nanostructures with surrogate fitness - target “worm”

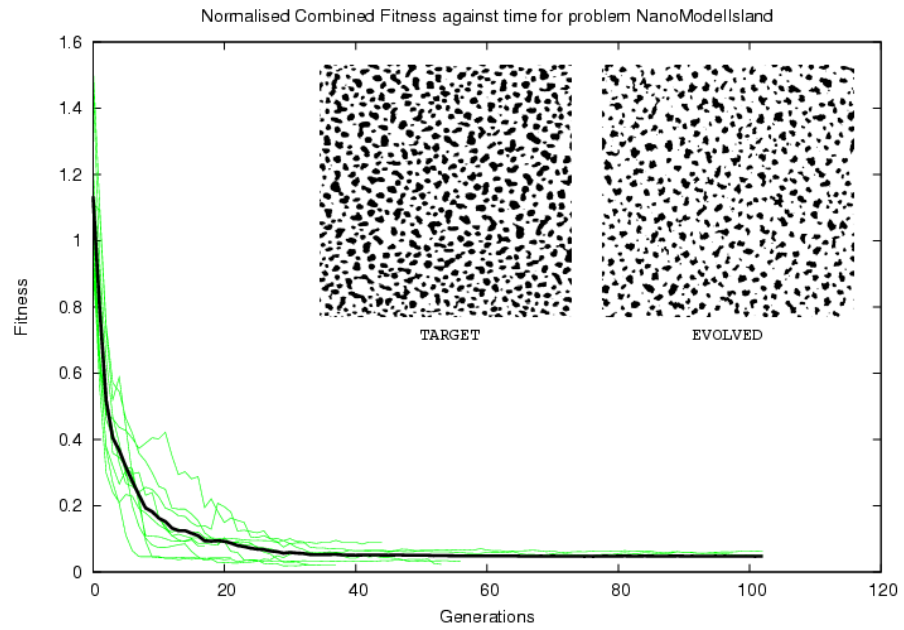


FIGURE 7.8: Evolution of nanostructures with surrogate fitness - target “island”

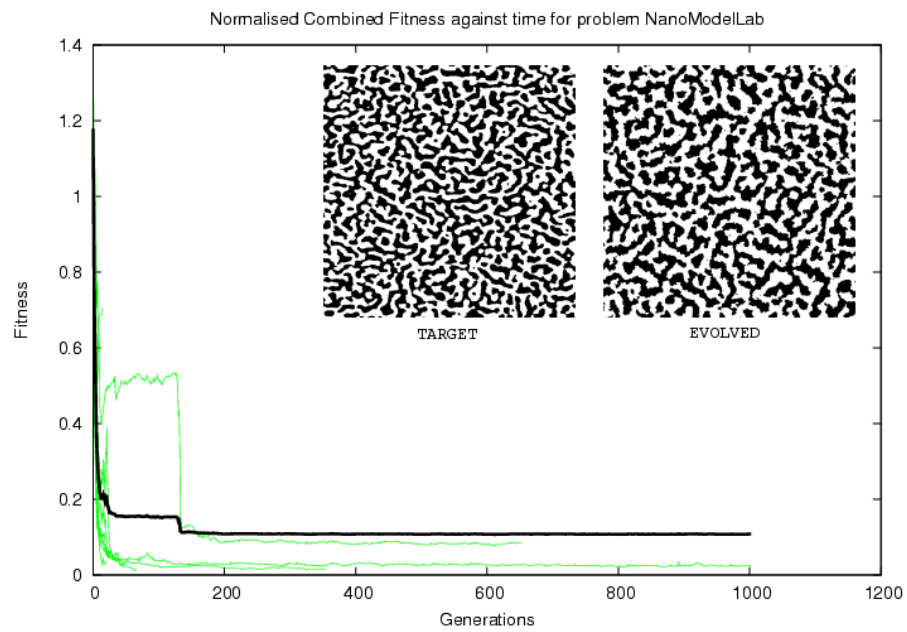


FIGURE 7.9: Evolution of nanostructures with surrogate fitness - target “labyrinth”

Target	Run	Generations	NCF	Resultant genotype (ADM)			
Cell	1	100	0.03052	18.0	0.2	0.632	2.047
Cell	2	17	0.07507	50.0	0.2	0.851	2.509
Cell	3	16	0.02774	16.0	0.2	0.906	2.905
Cell	4	21	0.02264	34.0	0.2	1.278	3.883
Cell	5	100	0.00347	20.0	0.2	0.839	2.668
Cell	6	21	0.04995	30.0	0.2	1.323	4.000
Cell	7	28	0.00668	18.0	0.2	1.023	3.213
Cell	8	17	0.02957	35.0	0.2	1.033	3.187
Cell	9	19	0.02764	22.0	0.2	1.306	4.000
Cell	10	23	0.01360	18.0	0.2	1.202	3.732
	$\mu$	36.2	0.02869				
	$c_v$	0.93387	0.02107				
Worm	1	95	0.04435	50.0	0.2	1.220	3.187
Worm	2	12	0.17175	35.0	0.2	1.194	3.065
Worm	3	75	0.07590	50.0	0.2	1.342	3.582
Worm	4	18	0.03282	50.0	0.2	0.915	2.403
Worm	5	39	0.16530	5.0	0.2	0.938	2.711
Worm	6	19	0.09312	26.0	0.2	0.956	2.597
Worm	7	17	0.09069	32.0	0.2	1.148	3.101
Worm	8	31	0.08623	50.0	0.2	1.170	3.140
Worm	9	20	0.03324	23.0	0.2	0.659	1.697
Worm	10	89	0.02921	46.0	0.2	0.866	2.255
	$\mu$	41.5	0.08226				
	$c_v$	0.77661	0.05191				
Island	1	100	0.06168	24.0	0.2	1.087	1.441
Island	2	32	0.02157	31.0	0.2	1.323	1.656
Island	3	36	0.02087	11.0	0.2	1.497	1.964
Island	4	55	0.03334	6.0	0.2	1.849	2.388
Island	5	20	0.04300	13.0	0.2	1.422	1.857
Island	6	38	0.05138	6.0	0.1	1.910	2.450
Island	7	43	0.09038	41.0	0.2	1.518	1.840
Island	8	16	0.07852	31.0	0.1	1.626	1.983
Island	9	100	0.04942	28.0	0.2	1.111	1.446
Island	10	52	0.02313	6.0	0.1	1.954	2.495
	$\mu$	49.2	0.04733				
	$c_v$	0.77661	0.02408				
Labyrinth	1	39	0.04375	26.0	0.3	1.480	1.734
Labyrinth	2	66	0.01295	26.0	0.3	1.522	1.784
Labyrinth	3	14	0.70365	24.0	0.3	1.808	2.108
Labyrinth	4	51	0.02774	8.0	0.3	1.632	2.009
Labyrinth	5	21	0.02958	32.0	0.3	1.775	2.032
Labyrinth	6	100	0.02516	27.0	0.3	1.759	2.027
Labyrinth	7	36	0.04468	5.0	0.3	1.835	2.243
Labyrinth	8	100	0.08488	18.0	0.3	0.789	1.023
Labyrinth	9	24	0.09001	27.0	0.3	1.535	1.774
Labyrinth	10	100	0.01615	28.0	0.3	1.493	1.751
	$\mu$	55.1	0.10786				
	$c_v$	0.62363	0.21101				

TABLE 7.3: Statistical analysis of surrogate-assisted nano system evolutions. The table also shows the evolved genotype, as chosen by the Automated Decision Maker, along with the associated Normalised Combined Fitness value.

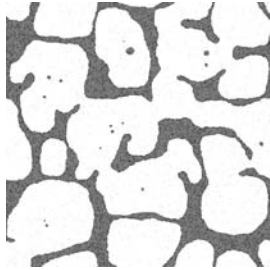
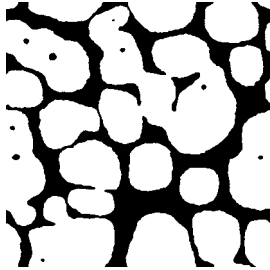
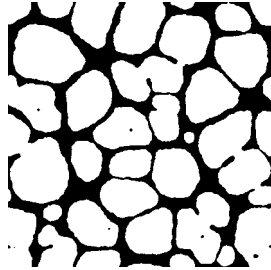
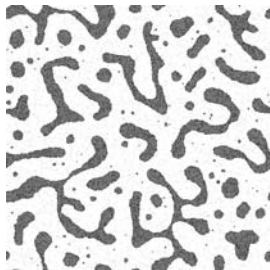
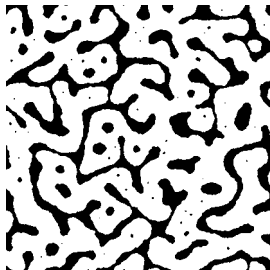
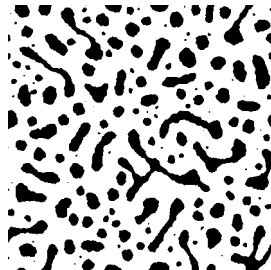
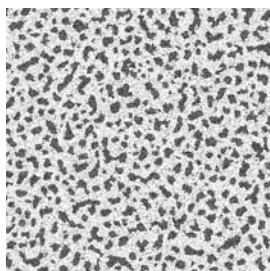
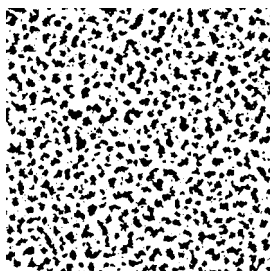
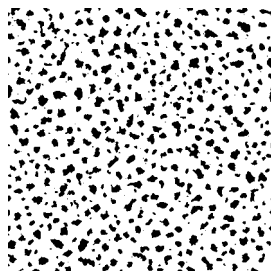
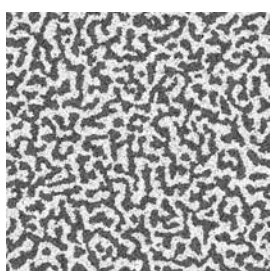
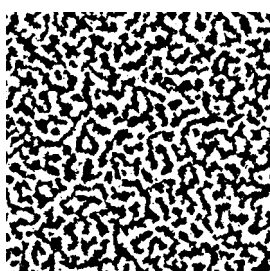
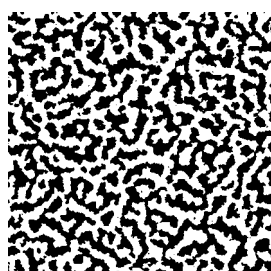
Target	Simulator + Minkowski		Neural Net model	
	Evolved	Time	Evolved	Time
		25:28:09 09:29:25 14:08:36 42:41:19 57:49:09 19:48:15 05:07:12 12:13:13 02:52:44 11:59:33		00:00:03 00:00:01 00:00:02 00:00:02 00:00:02 00:00:03 00:00:02 00:00:02 00:00:01 00:00:02
		25:33:29 06:19:19 05:59:35 60:58:24 07:14:33 22:30:28 08:27:01 01:46:41 02:45:28 11:22:17		00:00:03 00:00:03 00:00:02 00:00:02 00:00:02 00:00:02 00:00:02 00:00:02 00:00:02 00:00:02
		08:08:23 09:19:45 03:40:37 18:22:10 05:22:56 03:17:58 06:20:48 01:34:51 12:39:10 02:11:02		00:00:03 00:00:03 00:00:03 00:00:03 00:00:03 00:00:03 00:00:03 00:00:03 00:00:02 00:00:03
		09:19:45 102:26:55 02:42:17 38:20:19 18:21:49 14:22:11 04:09:37 02:57:13 02:14:04 03:16:47		00:00:03 00:00:03 00:00:03 00:00:03 00:00:03 00:00:03 00:00:04 00:00:03 00:00:03 00:00:03

FIGURE 7.10: Comparison of evolved patterns using a) the nano system simulator with Minkowski analysis and b) the neural network fitness approximation, along with total run-time statistics (hr:min:sec)

It is important to note that these results were obtained using *only* the approximation model. Much work of a similar nature combines the ‘real’ system alongside the surrogate model in a rather closer fashion, and a closer style of integration as part of the methodology presented above could form the basis of future research in this area. Indeed, the potential for ‘active learning’, i.e., training the approximation model at the same time as running the ‘real’ system, as in [30] is a particularly interesting notion.

## CHAPTER 8

# Conclusions and outlook

At the outset of this thesis, the hypothesis was posited that evolutionary algorithms can be employed to coerce complex chemical systems – more specifically, self-organised systems – into a pre-determined target behaviour.

In investigating this proposal, a toolbox of machine learning techniques was developed. The Evolutionary Engine presented in chapter 3 aims to be particularly user-friendly, easily configurable, flexible and ‘connectable’. We believe that the use of evolutionary computation techniques will give rise to a new, previously unreachable area of chemical functionality, whilst also giving insight into the underlying chemistry behind them. During the presentation of the Evolutionary Engine, a brief, but powerful illustration was given of its potential to be connected to a physical laboratory reactor array system, giving rise to the potential for real-time, optimisation of complex chemical processes. This level of sophistication requires the solution of a number of significant technical chemical control issues, and thus the majority of this thesis has presented work where the Evolutionary Engine has been coupled together with digital, simulated complex systems. We believe that the software developed and insights gained into these systems during the course of this research will be invaluable when such systems are to be used with real chemical reactors.

In chapter 4, it was observed that the nature of complex systems means that the mapping between the genotype that specifies a behaviour and the actual realisation of that behaviour – the phenotype – may not be one-to-one, and indeed may be highly non-linear, counter-intuitive and even stochastic. This relationship, and indeed that



between the phenotype and the numerical fitness attached to it by the objective function, is therefore of prime importance if an evolutionary algorithm, or for that matter any optimisation methodology, is to be a successful search method. Verifying whether an evolutionary algorithm is an effective method of optimisation for a given application using a given fitness function is a very difficult problem; chapter 4 presented a two-stage verification protocol which can give very useful insights into the potential (or otherwise) of the efficacy of a given representation and corresponding objective function when confronted with a complex genotype–phenotype–fitness mapping.

In chapter 5, the Evolutionary Engine’s ability to successfully coerce a cellular automaton-based system into a pre-defined, target behaviour was shown. Moreover, it was also shown that the results obtained from the genetic algorithm were superior to those of two standard non-evolutionary alternatives, adding to the weight of literature suggesting that an evolutionary methodology is better suited to this area of complex system design. The Universal Similarity Metric was shown to be particularly effective in driving the search for target behaviours in such a number of CA-based systems.

In chapter 6, for the first time, evolutionary algorithms were applied to a Monte Carlo model that simulates the self-organisation processes of spin cast nanoparticles. Notwithstanding the success of the USM in the evolving target behaviours in the CA-based systems, some shortcomings were observed, and hence the Minkowski functionals were identified as a simple but very effective new similarity measure. The results showed that this Minkowski-based metric in the context of a multi-objective genetic algorithm is a highly successful methodology in this novel problem domain. The mapping from simulator parameters onto those of the actual laboratory equipment is not a straight-forward one, granted, but the methods developed in this part of the thesis and the corresponding results would seem to suggest that such a methodology could be well suited to the optimisation of a real physio-chemical system – as demonstrated in chapter 3.3, the Evolutionary Engine can be connected directly to suitable laboratory apparatus.

As identified at the outset of this thesis, a lengthy run time and expensive computation are both symptoms of many complex system problems. A further hypothesis was proposed that such systems can be modelled by a simpler approximation, result-

ing in savings of time and computation expense without compromising the quality of results. Chapter 7 showed that indeed, a neural network ensemble can accurately embody not only the behaviour of the nano simulator complex system, but also its subsequent mapping onto the Minkowski analysis methods. This innovative approximation method results in a dramatic saving of time with little compromise to the quality of results, even using only the approximation model. A closer integration of real and approximated systems was identified as a possible direction for future research, in particular the potential for an ‘active learning’ environment where the approximation model is trained at the same time as running the ‘real’ system.

\*

Self-organisation is a relatively new area of research – there is much still to be discovered, much still to be formalised. A greater understanding of the processes involved in dynamic self-assembly systems could herald the start of a revolutionary new paradigm of evolvable chemical complexity, and it is hoped that the research presented in this thesis has contributed towards this goal.

Evolutionary computation methods are becoming ever more important in today’s engineering research environment – not only is rational design of such systems often infeasible due to the size of the search space, but the complex, non-linear and counter-intuitive nature of the systems that today’s scientists are wanting to develop is such that a rational, human approach can be considerably limited in its success.

As mentioned at the outset of this thesis, the central aim of this research has been to address some of the issues involved in the interpretation of complex behaviour such that it is suitable for optimisation (by whatever method). Armed with the selection of techniques developed in this thesis, a natural extension to this part of the research is, therefore, to switch the focus to the meta-heuristic algorithms themselves, surveying the enormous range of techniques developed for large, complex and noisy search spaces, such that the results presented here can be improved upon and/or obtained in a more timely fashion. The work presented in chapter 7 is therefore of particular significance, showing that approximations of complex systems can be used to good effect in reducing the running time of these otherwise very costly experiments.

In addition, a more sophisticated range of meta-heuristics may become a necessity if higher dimension problems are considered – those presented above have been in no more than eight dimensions (and generally three or four) but, as mentioned at the very outset of this thesis, it is not uncommon for chemical systems to involve thirty or more input parameters.

The question posed by the authors of [44] – “how do we build artificial systems (or manage natural ones) so that the properties that emerge are the ones we want” – has been the central motivation behind the work described in this dissertation. Specifically, the work has aimed to address some of the problems involved in interpreting complex behaviour such that it is suitable (and robust) for optimisation. In particular, the application of evolutionary computation to design of nanostructured systems is both a novel and a significant contribution to the field of complex systems design. The potential of this work to be extended to the construction of real nanoscale components as part of an ‘unconventional computing’ system is particularly relevant in today’s research environment, and indeed, there is a clear analogy between the behaviour of such a system and the origins of biological life. If the computational methods used to evolve target behaviours in such nano-scale systems could be applied to a physical system, rather than a simulation, the potential for a ‘dial a function’ style of component design (recently identified as an EPSRC ‘Grand Challenge’ [1]) is exciting indeed. This is the natural extension to the work presented above. Much of the burden falls on the engineers – developing a computer-controllable laboratory setup is non-trivial in the extreme, though, as discussed below, initial work in this area is encouraging, and suggests that the ability to perform such *in vitro* optimisation is close at hand.

Initial results shown at the outset of this thesis demonstrated the capabilities of the Evolutionary Engine when coupled, via a remote link, to a computer controlled chemical reactor array. This achievement, coupled with the algorithms and techniques discussed during the course of this thesis for both evolving complex behaviour, and verifying the robustness of such techniques suggest that the artificial evolution of life-like features in complex chemical systems is not far off. The aim of this thesis was to provide a ‘stepping stone’ towards this lofty goal; the work presented in the chapters

above, and in the various publications that have stemmed from this research has, I hope, provided such a stepping stone, and contributed to this ever growing and ever more important area of scientific research.

## References

- [1] Chemical sciences and engineering grand challenges: Report of workshop outcomes. <http://www.epsrc.ac.uk/CMSWeb/Downloads/Other/grandchallengesreport.pdf>, July 2009.
- [2] L Altenberg. Fitness distance correlation analysis: An instructive counterexample. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.
- [3] T Back, R Breukelaar, and L Willmes. Inverse Design of Cellular Automata by Genetic Algorithms: An Unconventional Programming Paradigm. *Lecture notes in Computer Science*, 3566:161, 2005.
- [4] L Barone, L While, and P Hingston. Designing crushers with a multi-objective evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.
- [5] D Barthel, JD Hirst, J Blazewicz, EK Burke, and N Krasnogor. Procksi: a decision support system for protein (structure) comparison, knowledge, similarity and information. *BMC Bioinformatics*, 8, 2007.
- [6] TP Bigioni, XM Lin, TT Nguyen, EI Corwin, TA Witten, and HM Jaeger. *Nature Mat.*, 5, 2006.
- [7] J Biles, P Anderson, and L Loggi. Neural network fitness functions for a musical iga. In *Proceedings of the International ICSC Symposium on Intelligent Industrial Automation (IIA'96)*. International Computing Sciences Conferences (ICSC), 1996.

- [8] KJM Bishop, CJ Campbell, G Mahmud, and BA Grzybowski. *Biomimetic Design of Dynamic Self-Assembling Systems*. Elsevier, 2008.
- [9] MO Blunt, CP Martin, M Ahola-Tuomi, E Pauliac-Vaujour, P Sharp, P Nativio, M Brust, and P Moriarty. Coerced mechanical coarsening of nanoparticle assemblies. *Nature Nanotechnology*, pages 167 – 170, 2007.
- [10] AM Bouchard, CE Warrender, and GC Osbourn. *The Programming language of dynamic self-assembly*. Elsevier, 2008.
- [11] R. Breukelaar and T Back. Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 107–114, New York, NY, USA, 2005. ACM Press.
- [12] SD Brown. *Field-programmable Gate Arrays*. Springer, 1993.
- [13] Y Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Computer Science*, 378(1), 2007.
- [14] J Brzustowski. qclust v0.2. <http://www.biology.ualberta.ca/jbrzusto>, 1998.
- [15] L Bull, A Budd, C Stone, I Uroukov, B de Lacy Costello, and A Adamatzky. Towards unconventional computing through simulated evolution: Control of nonlinear media by a learning classifier system. *Artificial Life*, 14(2):203–222, 2008.
- [16] L Bull, I Lawson, A Adamatzky, and B DeLacyCostello. Towards predicting spatial complexity: A learning classifier system approach to cellular automata identification. In *In Proceedings of the IEEE Congress on Evolutionary Computation*, pages 136–141, 2005.
- [17] E Cantupaz and C Kamath. Evolving neural networks to identify bent-double galaxies in the first survey. *Neural Networks*, 16(3-4):507–517, 2003.

- [18] L Cardelli and P Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.
- [19] B Chopard and M Droz. *Cellular automata modeling of physical systems*. Cambridge University Press, 1998.
- [20] CA Coello Coello. Evolutionary multi-objective optimisation: An historical view of the field. *Computational Intelligence Magazine*, 1(1):28–36, 2006.
- [21] CA Coello Coello, DA van Veldhuizen, and GB Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.
- [22] L Cronin, N Krasnogor, BG Davis, C Alexander, N Robertson, JHG Steinke, SLM Schroeder, AN Khlobystov, G Cooper, P Gardner, and P Siepmann. Is it alive? recognising cellular systems: A computational-chemical perspective. *Nature: Biotechnology*, 24:1203 – 1206, 2006.
- [23] C Darwin. *On The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.
- [24] C Darwin and AR Wallace. *On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection*. Linnean Society of London, 1858.
- [25] R Dawkins. *The Extended Phenotype: The Long Reach of the Gene*. Oxford University Press, 1982.
- [26] R Dawkins. *The Blind Watchmaker*. Norton, 1996.
- [27] R Dawkins. *Climbing Mount Improbable*. Penguin Books, 1996.
- [28] Pablo Moisset de Espanes. *Computer Aided Search for Optimal Self-Assembly Systems*. Elsevier, 2008.
- [29] KA de Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

- [30] K Deb and P Nain. *An Evolutionary Multi-objective Adaptive Meta-modeling Procedure Using Artificial Neural Networks*. Springer, 2007.
- [31] K Deb, A Pratap, S Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions in Evolutionary Computation*, 6(2):182–197, 2002.
- [32] P Dittrich, J Ziegler, and W Banzhaf. Artificial chemistries - a review. *Artificial Life*, 7(3):225–275, 2001.
- [33] DP DiVincenzo. Quantum computation. *Science*, 270(5234):255–261, 1995.
- [34] TB Downing. *Java RMI: Remote Method Invocation*. Wiley, 1998.
- [35] A Eiben and J Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
- [36] L Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*. Prentice-Hall, 1994.
- [37] K-C Fu, Y Zhai, and S Zhou. Optimum design of welded steel plate girder bridges using a genetic algorithm with elitism. *Journal of Bridge Engineering*, 10(3):291–301, 2005.
- [38] M Gardner. The fantastic combinations of john conway’s new solitary game of ”life”. *Scientific American*, 223:120–123, 1970.
- [39] G Ge and L Brus. *J. Phys. Chem. B*, 104, 2000.
- [40] M Gheorghe and G Paun. *Computing by Self-Assembly: DNA Molecules, Polyminoes, Cells*. Elsevier, 2008.
- [41] DE Goldberg and K Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996.
- [42] D Goswami. Optical computing. *Resonance*, 8(6), 2003.
- [43] D Graham-Rowe. Introducing the glooper computer. *New Scientist*, 185(2492):32–36, 2005.



- [44] G Green and D Newat. Towards a theory of everything? - grand challenges in complexity and informatics. *Complexity International*, 8, 2001.
- [45] GW Greenwood and AM Tyrrell. *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley-IEEE Press, 2006.
- [46] Y Guo, G Poulton, C Murray, and G James. Designing self-assembly dna-based structures in a multi-agent system. In *Proceedings of the 7th Asia-Pacific Conference on Complex Systems*, pages 637–648, 2004.
- [47] M Halkidi, Y Batistakis, and M Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2):107–145, 2001.
- [48] S Harding, J Miller, and E Rietman. Evolution in materio: Evolving logic gates in liquid crystal. In *European Conference on Artificial Life: Workshop on Unconventional Computing*, 2005.
- [49] S Harding, J Miller, and E Rietman. Evolution in materio: Exploiting the physics of materials for computation. *IEEE Transactions on Nanotechnology*, 2005.
- [50] D Harel. A turing-like test for biological modeling. *Nature Biotechnology*, 23(4):495–496, 2005.
- [51] R Hecht-Nielsen. Theory of the backpropagation neural network. In *Proceedings of the International Joint Conference on Neural Networks*, pages 593–605, 1989.
- [52] R Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *IEEE International Conference on Evolutionary Computation*, volume 1, 1995.
- [53] R Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 384–, 1995.

- [54] RL Johnston HM Cartwright. *Applications of evolutionary computation in chemistry*. Springer, 2004.
- [55] SY Ho, LS Shu, and JH Chen. Intelligent evolutionary algorithms for large parameter optimization problems. *IEEE Transactions on Evolutionary Computation*, 8(6):522–541, 2004.
- [56] JH Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.
- [57] J Horn. Coevolving species for shape nesting. In *Proceedings of IEEE Congress on Evolutionary Computation*, 2005.
- [58] G Hornby and J Pollack. Evolving l-systems to generate virtual creatures. *Computers and Graphics*, 25:1041–1048, 2001.
- [59] K Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.
- [60] K Hornik, M Stinchcombe, and H White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 1989.
- [61] Y Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.
- [62] Y Jin and B Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In *Genetic and Evolutionary Computation (GECCO 2004)*, pages 688–699, 2004.
- [63] T Jones and S Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, 1995.

- [64] JB Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.
- [65] M Keijzer, JJ Merelo, G Romero, and M Schoenauer. Evolving objects: a general purpose evolutionary computation library. In *Proceedings of the Fifth International Conference on Artificial Evolution*, 2001.
- [66] L Kier, P Seybold, and C Cheng. *Cellular Automata Modeling of Chemical Systems*. Springer, 2005.
- [67] H Kita and Y Sano. Genetic algorithms for optimisation of noisy fitness functions and adaption to changing environments.
- [68] J Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.
- [69] N Krasnogor and D Pelta. Measuring the similarity of protein structures by means of the universal similarity metric. *Bioinformatics*, 20(7):1015–1021.
- [70] A Lazcano. *The transition from living to non-living*, pages 60–69. Columbia University Press, 1994.
- [71] RE Lensk, C Ofria, RT Pennock, and C Adami. The evolutionary origin of complex features. *Nature*, 423(6936):139–44, 2003.
- [72] M Li, X Chen, X Li, B Ma, and P Vitnyi. The similarity metric. In *Proceedings of the Fourteenth Annual ACM-SIAM symposium on discrete algorithms*, pages 863–872, 2003.
- [73] M Li and P Vit. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

- [74] M Ling and R Sharp. Melody classification using a similarity metric based on kolmogorov complexity. In *Proceedings of the Conference on Sound and Music Computing*, 2004.
- [75] JD Lohn, GS Hornby, and DS Linden. An evolved antenna for deployment on nasa's space technology 5 mission. In *Proceedings of the Genetic Programming Theory Practice Workshop*, 2004.
- [76] S Luke, L Panait, G Balan, S Paus, Z Skolicki, J Bassett, R Hubley, and A Chircop. Ecj: a java based evolutionary computation research system. 2006.
- [77] N Marchettini, S Ristori, F Rossi, and M Rustici. An experimental model for mimicking biological systems: the belousov-zhabotinsky reaction in lipid membranes. *International Journal of Ecodynamics*, 1(1):55–63, 2006.
- [78] SJ Marrink and AE Mark. Molecular dynamics simulation of the formation, structure, and dynamics of small phospholipid vesicles. *Journal of the American Chemical Society*, 125(49):15233–15242, 2003.
- [79] P Marrone. Joone. <http://joone.sourceforge.net>.
- [80] CP Martin, MO Blunt, and P Moriarty. *Nano Lett.*, 4, 2004.
- [81] CP Martin, MO Blunt, and P Moriarty. Nanoparticle networks on silicon: self-organized or disorganized? *Nano Letters*, 4:271–274, 2004.
- [82] WS McCulloch and W Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- [83] Z Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.
- [84] K Michielsen and H de Raedt. Integral-geometry morphological image analysis. *Physics Reports*, 347:461–538, 2001.
- [85] BL Miller and DE Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, pages 193–212, 1995.

- [86] M Mitchell, J Crutchfield, and R Das. Evolving cellular automata with genetic algorithms: A review of recent work. In *Proceedings of the First International Conference on Evolutionary Computation and its Applications*, 1996.
- [87] C Moles, P Mendes, and J Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Research*, 13:2467–2474, 2003.
- [88] P Moriarty, MDR Taylor, and M Brust. *Phys. Rev. Lett*, 89, 2002.
- [89] S Narayanan, J Wang, and XM Lin. *Phys. Rev. Lett.*, 93, 2004.
- [90] H Ng, D Lim, Y Ong, B Lee, L Freund, S Parvez, and B Sendhoff. A multi-cluster grid enabled evolution framework for aerodynamic airfoil design optimization. In *Proceedings of the International Conference on Natural Computation*, pages 1112–1121, 2005.
- [91] J Oliver and R Perry. Definitely life but not definitively. *Origins of Life and Evolution of the Biosphere*, 36(5-6):515–521, 2006.
- [92] M Palesi and T Givargis. Multi-objective design space exploration using genetic algorithms. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, 2002.
- [93] G Palyi, C Zucchi, and L Caglioti. *Fundamentals of life*. Elsevier, 2002.
- [94] R Penrose. *The Emperor’s New Mind: Concerning Computers, Minds and The Laws of Physics*. Oxford University Press, 1989.
- [95] RS Perry and VM Kolb. The importance of chemicals from the transition zone to chemical evolution. In *Proceedings of the Third European Workshop on Exo-Astrobiology. Mars: The Search for life*, 2004.
- [96] AD Pietro, L While, and L Barone. Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1254–1261, 2004.

- [97] M Powner, B Gerland, and J Sutherland. Synthesis of activated pyrimidine ribonucleotides in prebiotically plausible conditions. *Nature*, 459:239–242, May 2009.
- [98] E Rabani, DR Reichman, PL Geissler, and LE Brus. Drying-mediated self-assembly of nanoparticles. *Nature*, 426:271–274, 2003.
- [99] J Rieffel and J Pollack. Evolutionary fabrication: the emergence of novel assembly methods in artificial ontogenies. In *Proceedings of the Genetic and Evolutionary Computation Conference: SEEDS workshop*, 2005.
- [100] J Rieffel and JB Pollack. Crossing the fabrication gap: Evolving assembly plans to build 3-d objects. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.
- [101] J Riskin. The defecating duck, or, the ambiguous origins of artificial life. *Critical Inquiry*, 29:599–633, 2003.
- [102] M Rodriguez-Fernandez, JA Egea, and JR Banga. Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. *BMC Bioinformatics*, 7, 2006.
- [103] FJ Romero-Campero, H Cao, M Camara, and N Krasnogor. Structure and parameter estimation for cell systems biology models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2008.
- [104] FJ Romero-Campero, J Twycross, H Cao, J Blakes, and N Krasnogor. A multiscale modeling framework based on p systems. In *Proceedings of the 9th International Workshop on Membrane Computing*, pages 63–77, 2008.
- [105] PWK Rothmund. Using lateral capillary forces to compute by self-assembly. In *Proceedings of the National Academy of Science, USA*, volume 47, pages 984–989, 2000.

- [106] O Roudenko and M Schoenauer. A steady performance stopping criterion for pareto-based evolutionary algorithms. In *Proceedings of the 6th International Multi-Objective Programming and Goal Programming Conference*, 2004.
- [107] E Sapin, O Bailleux, and J Chabrier. Research of complex forms in the cellular automata by evolutionary algorithms. In *Proceedings of Artificial Evolution: Sixth International Conference*, volume 2936, pages 357–367, 2003.
- [108] Emmanuel Sapin, Olivier Bailleux, Jean-Jacques Chabrier, and Pierre Collet. A new universal cellular automaton discovered by evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 175–187, 2004.
- [109] M Schoenauer. Crossing the chasm between theory and practice in evolutionary algorithms. <http://guide.gforge.inria.fr>.
- [110] R Schulman and E Winfree. Self-replication and evolution of dna crystals. In *Proceedings of the 13th European Conference on Artificial Life*, pages 734–743. Springer, 2005.
- [111] JR Searle. Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3, 1980.
- [112] RR Selmic and FL Lewis. Neural-network approximation of piecewise continuous functions: application to friction compensation. *IEEE Transactions on Neural Networks*, 13(3):745–751, 2002.
- [113] D Shaw, J Miles, and A Gray. Designing geodesic domes using a computational geometry-based representation. In *Proceedings of the Seventh International Conference of Adaptive Computing in Design and Manufacture*, 2006.
- [114] PA Siepmann, G Terrazas, and N Krasnogor. Evolutionary design for the behaviour of cellular automaton-based complex systems. In *Proceedings of the Seventh International Conference of Adaptive Computing in Design and Manufacture*, 2006.

- [115] O Steinbock, P Kettunen, and K Showalter. Chemical wave logic gates. *Journal of Physical Chemistry*, 100:18970–18975, 1996.
- [116] G Sywerda. Uniform crossover in genetic algorithms. *Proceedings of the third international conference on Genetic algorithms table of contents*, pages 2–9, 1989.
- [117] B Tadic. *From Microscopic Rules to Emergent Cooperativity in Large-Scale Patterns*. Elsevier, 2008.
- [118] G Terrazas. *Automated Evolutionary Design of Self-Assembly and Self-Organising Systems*. PhD thesis, 2008.
- [119] G Terrazas, M Gheorghe, G Kendall, and N Krasnogor. Evolving tiles for automated self-assembly design. In *IEEE Congress on Evolutionary Computation*, 2007.
- [120] G Terrazas, PA Siepmann, G Kendall, and N Krasnogor. An evolutionary methodology for the automated design of cellular automaton-based complex systems. *Journal of Cellular Automata*, 2:77–102, 2007.
- [121] M Theis, G Gazzola, M Forlin, I Poli, MM Hanczyc, and MA Bedau. Optimal formulation of complex chemical systems with a genetic algorithm. *ComPlexUs*, page In press, 2007.
- [122] A Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *Proceedings of the First International Conference on Evolvable Systems*, 1996.
- [123] S Tisue and U Wilensky. Netlogo: A simple environment for modeling complexity. In *Proceedings of the International Conference on Complex Systems*, 2004.
- [124] T Toffoli and N Margolus. *Cellular automata machines - a new environment for modelling*. MIT press, 1987.

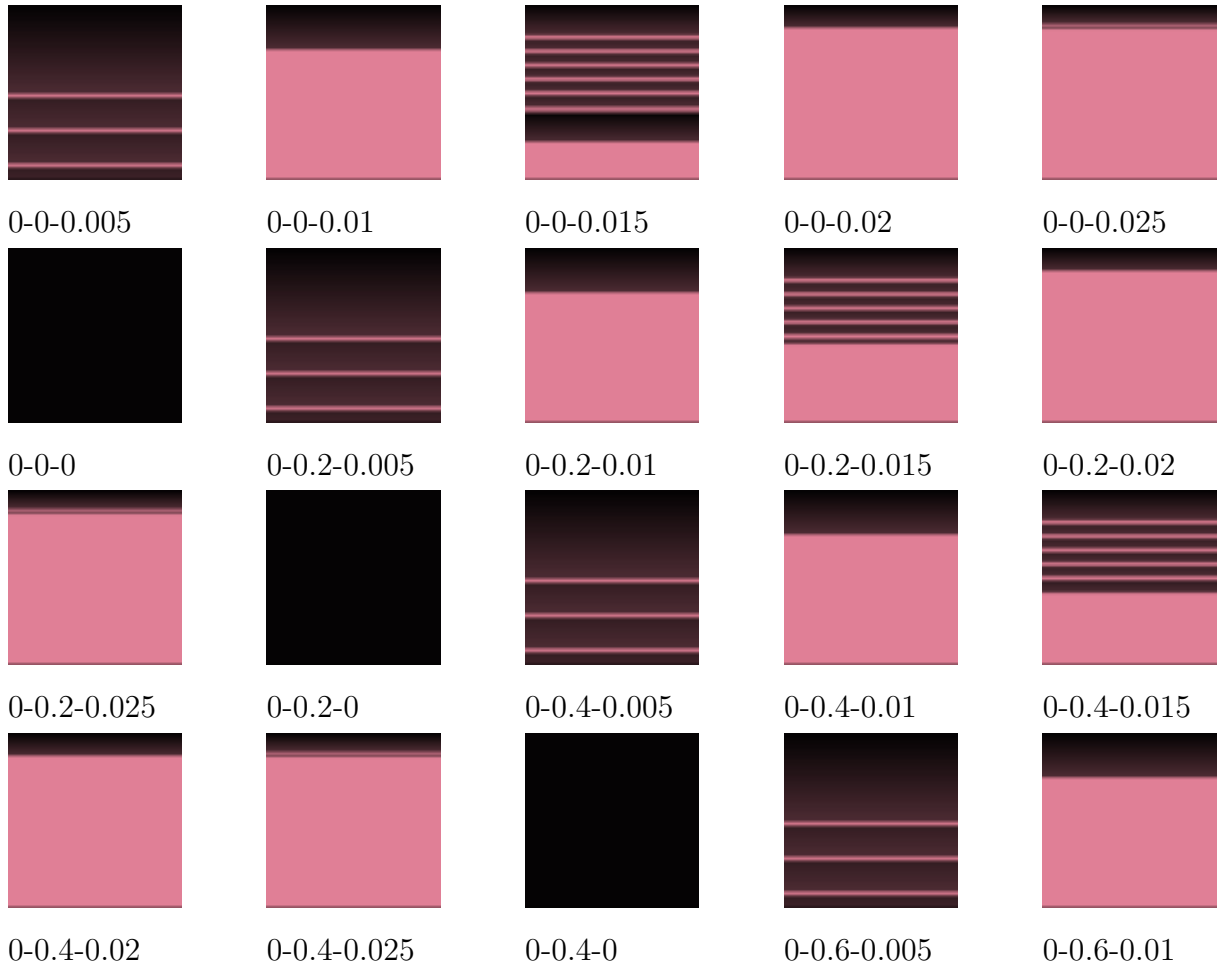


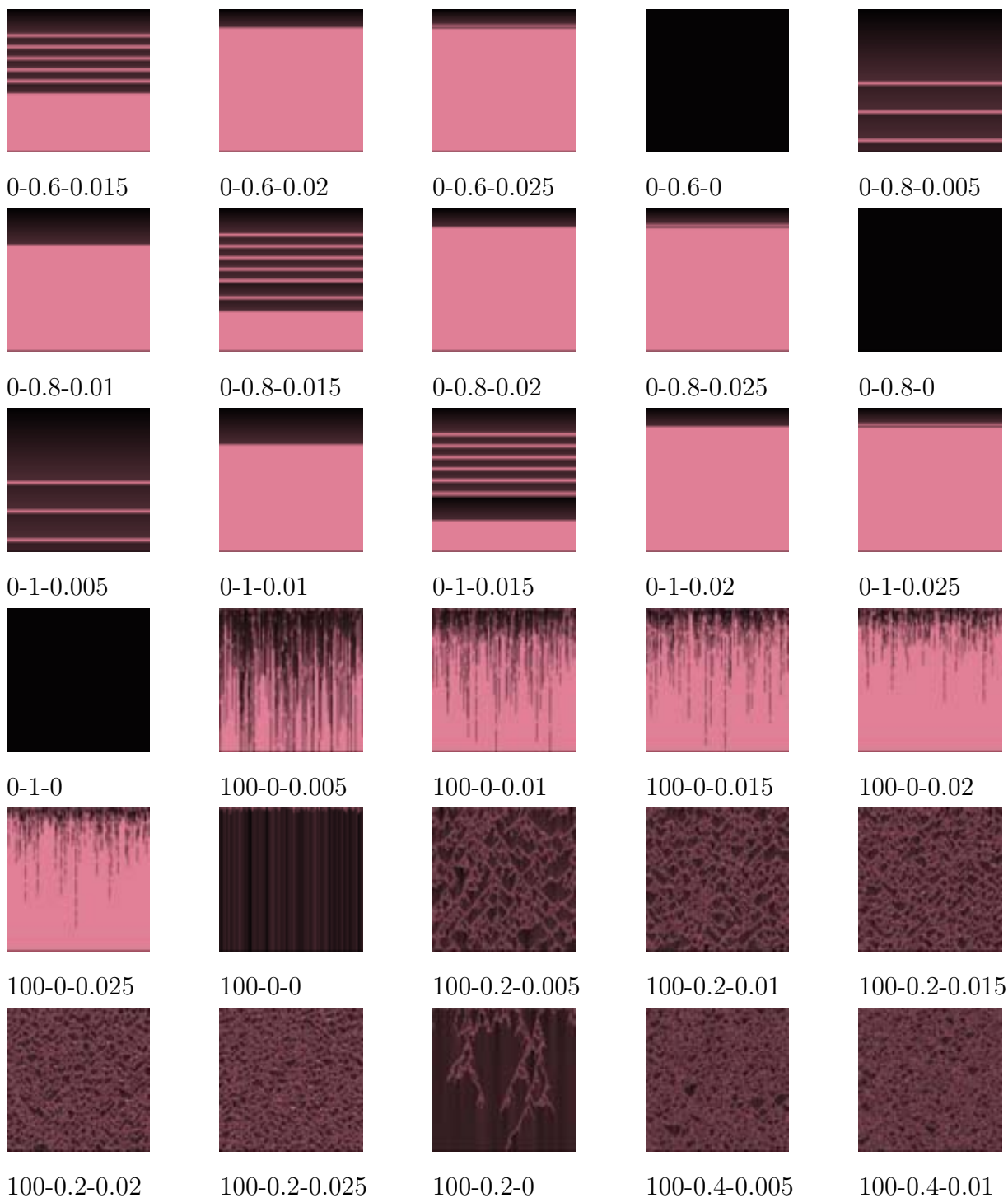
- [125] M Tomassini, L Vanneschi, P Collard, and M Clergue. A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 13(2).
- [126] J Tour, W van Zandt, C Husband, S Husband, L Wilson, P Franzon, and D Nackashi. Nanocell logic gates for molecular computing. *IEEE Transactions on Nanotechnology*, 1(2), 2002.
- [127] A Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.
- [128] L Vanneschi, M Tomassini, P Collard, and M Clergue. Fitness distance correlation in structural mutation genetic programming. In *Proceedings of EuroGP*, 2003.
- [129] L von Ahn, M Blum, NJ Hopper, and J Langford. *CAPTCHA: Using Hard AI Problems for Security*. 2003.
- [130] JR Weimar, JJ Tyson, and LT Watson. Third generation cellular automaton for modeling excitable media. *Physica D*, 55:328–339, 1992.
- [131] S Wolfram. *A New Kind of Science*. Wolfram Media, 2002.
- [132] DH Wolpert and WG Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.
- [133] VV Yashin and AC Balazs. Pattern formation and shape changes in self-oscillating polymer gels. *Science*, 314.

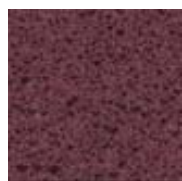
## APPENDIX A

# Datasets

### A.1 Turbulence







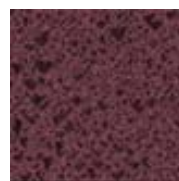
100-0.4-0.015



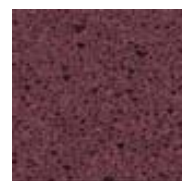
100-0.4-0.02



100-0.4-0.025



100-0.4-0



100-0.6-0.005



100-0.6-0.01



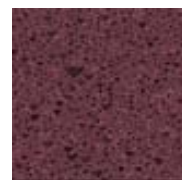
100-0.6-0.015



100-0.6-0.02



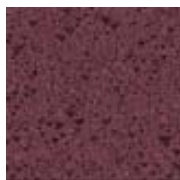
100-0.6-0.025



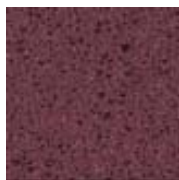
100-0.6-0



100-0.8-0.005



100-0.8-0.01



100-0.8-0.015



100-0.8-0.02



100-0.8-0.025



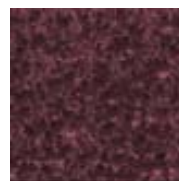
100-0.8-0



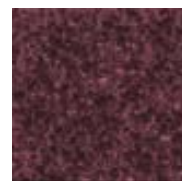
100-1-0.005



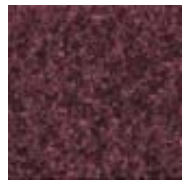
100-1-0.01



100-1-0.015



100-1-0.02



100-1-0.025



100-1-0



20-0-0.005



20-0-0.01



20-0-0.015



20-0-0.02



20-0-0.025



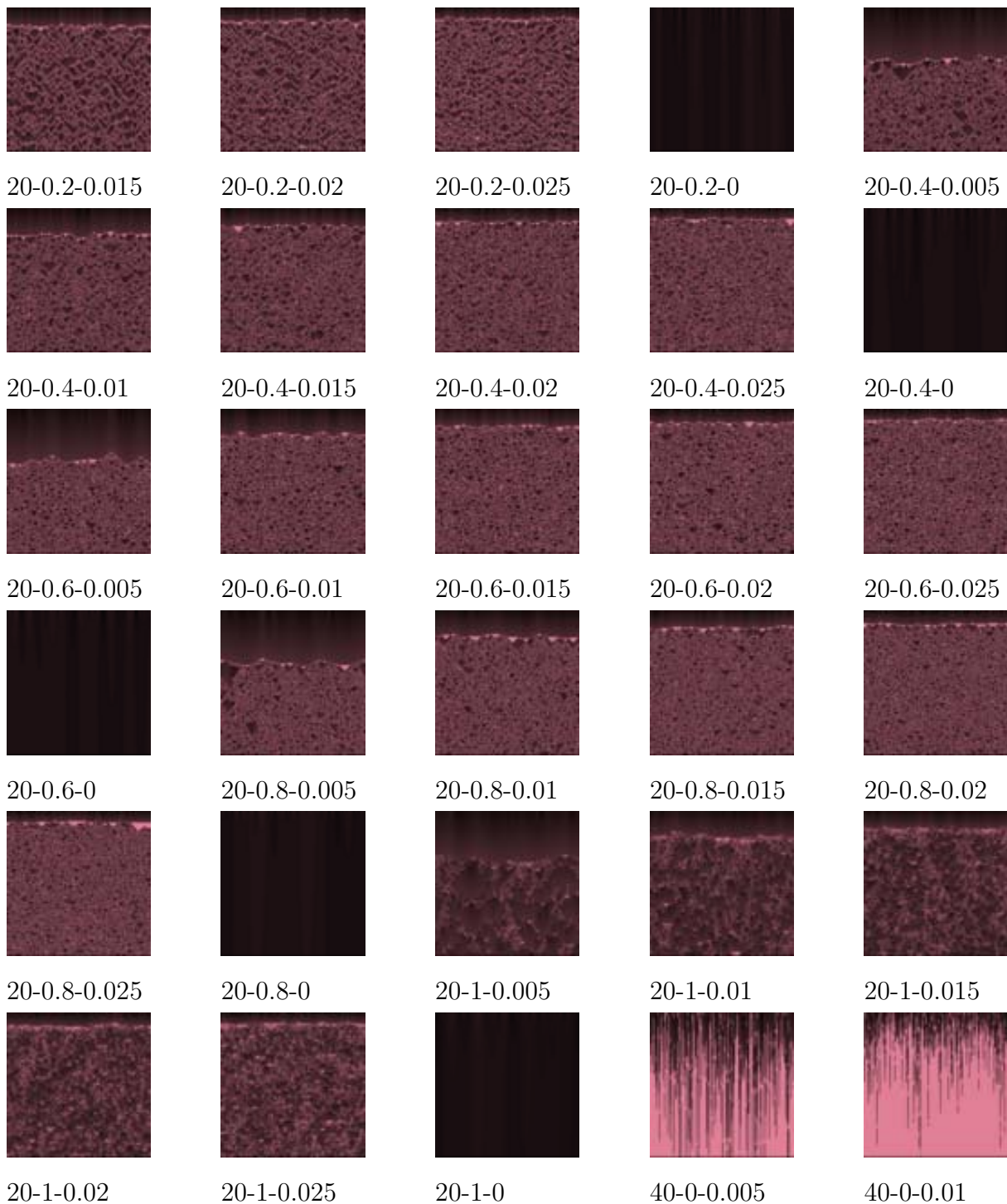
20-0-0



20-0.2-0.005



20-0.2-0.01





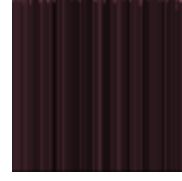
40-0-0.015



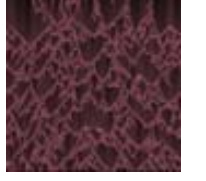
40-0-0.02



40-0-0.025



40-0-0



40-0.2-0.005



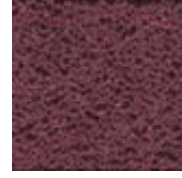
40-0.2-0.01



40-0.2-0.015



40-0.2-0.02



40-0.2-0.025



40-0.2-0



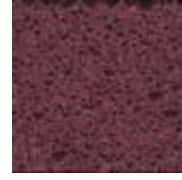
40-0.4-0.005



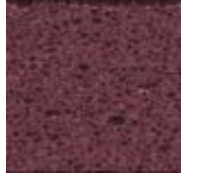
40-0.4-0.01



40-0.4-0.015



40-0.4-0.02



40-0.4-0.025



40-0.4-0



40-0.6-0.005



40-0.6-0.01



40-0.6-0.015



40-0.6-0.02



40-0.6-0.025



40-0.6-0



40-0.8-0.005



40-0.8-0.01



40-0.8-0.015



40-0.8-0.02



40-0.8-0.025



40-0.8-0

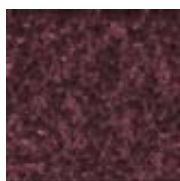


40-1-0.005



40-1-0.01





40-1-0.015



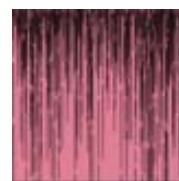
40-1-0.02



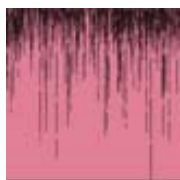
40-1-0.025



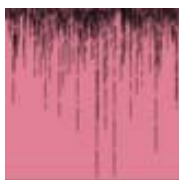
40-1-0



60-0-0.005



60-0-0.01



60-0-0.015



60-0-0.02



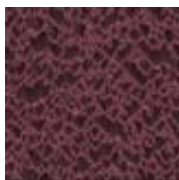
60-0-0.025



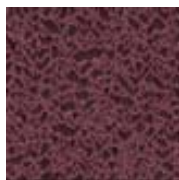
60-0-0



60-0.2-0.005



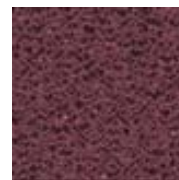
60-0.2-0.01



60-0.2-0.015



60-0.2-0.02



60-0.2-0.025



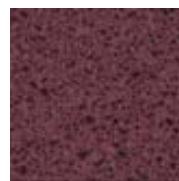
60-0.2-0



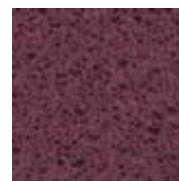
60-0.4-0.005



60-0.4-0.01



60-0.4-0.015



60-0.4-0.02



60-0.4-0.025



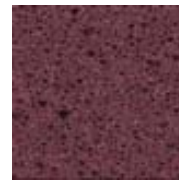
60-0.4-0



60-0.6-0.005



60-0.6-0.01



60-0.6-0.015



60-0.6-0.02



60-0.6-0.025



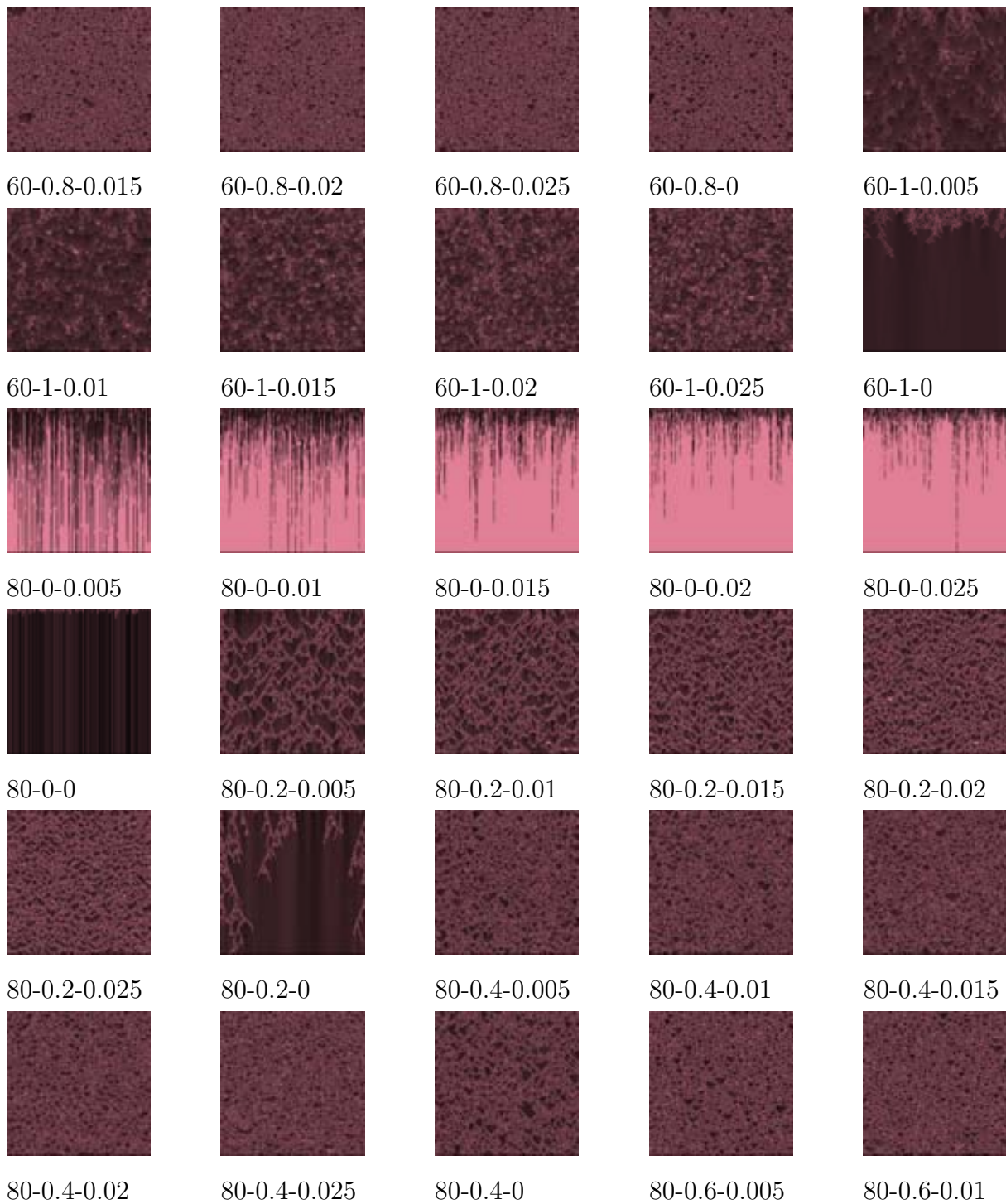
60-0.6-0



60-0.8-0.005



60-0.8-0.01







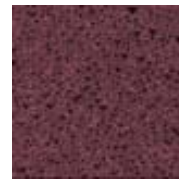
80-0.6-0.015



80-0.6-0.02



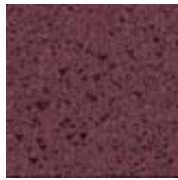
80-0.6-0.025



80-0.6-0



80-0.8-0.005



80-0.8-0.01



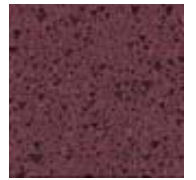
80-0.8-0.015



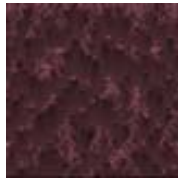
80-0.8-0.02



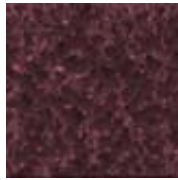
80-0.8-0.025



80-0.8-0



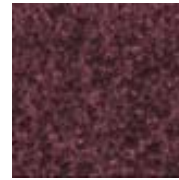
80-1-0.005



80-1-0.01



80-1-0.015



80-1-0.02

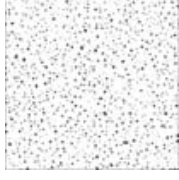


80-1-0.025



80-1-0

## A.2 Nanostructures

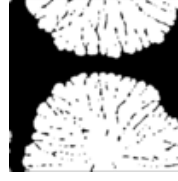


5-0.05-0.5-1



5-0.05-0.5-2

5-0.05-0.5-3

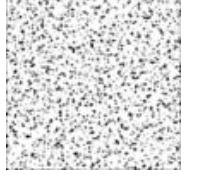


5-0.05-1-4

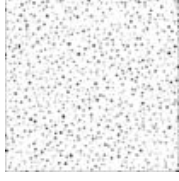
5-0.05-0.5-4



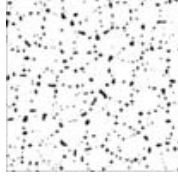
5-0.05-1.5-1



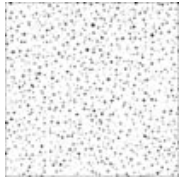
5-0.05-1-1



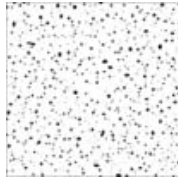
5-0.05-1-2



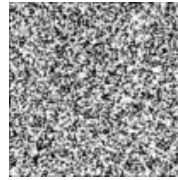
5-0.05-1-3



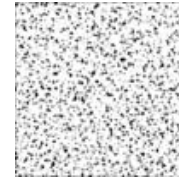
5-0.05-1.5-3



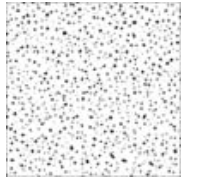
5-0.05-1.5-4



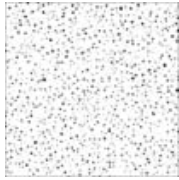
5-0.05-2-1



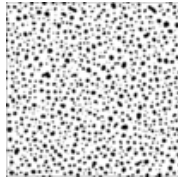
5-0.05-2-2



5-0.05-2-3

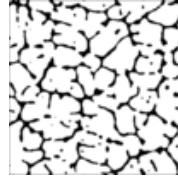


5-0.05-2-4



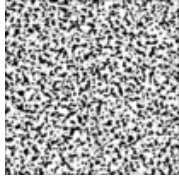
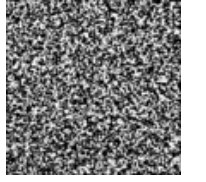
5-0.13-0.5-1

5-0.13-0.5-2

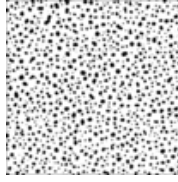


5-0.13-0.5-3

5-0.13-0.5-4

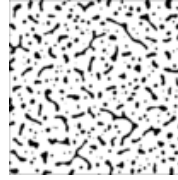


5-0.13-1-1



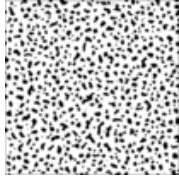
5-0.13-1-2

5-0.13-1-3

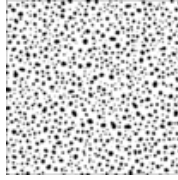


5-0.13-1-4

5-0.13-1.5-1

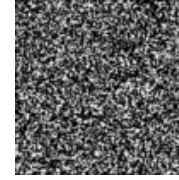


5-0.13-1.5-2

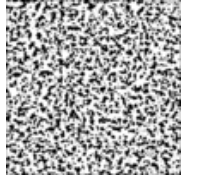


5-0.13-1.5-3

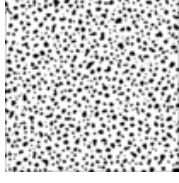
5-0.13-1.5-4



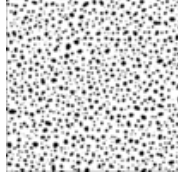
5-0.13-2-1



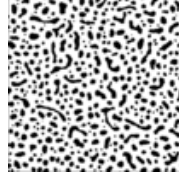
5-0.13-2-2



5-0.13-2-3



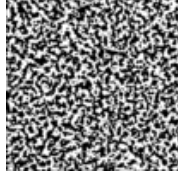
5-0.13-2-4



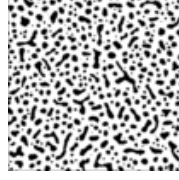
5-0.21-0.5-1

5-0.21-0.5-2

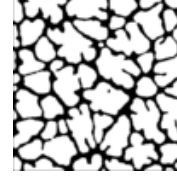
5-0.21-0.5-3



5-0.21-1-1



5-0.21-1-2



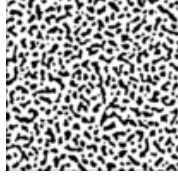
5-0.21-1-3

5-0.21-1-4

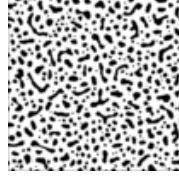
5-0.21-0.5-4



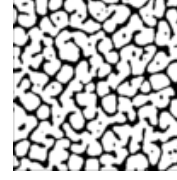
5-0.21-1.5-1



5-0.21-1.5-2

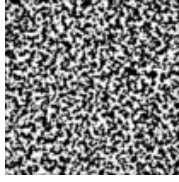


5-0.21-1.5-3

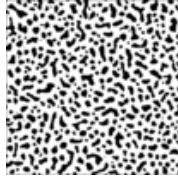


5-0.21-1.5-4

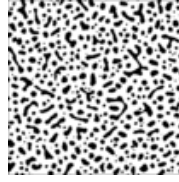
5-0.21-2-1



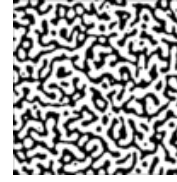
5-0.21-2-2



5-0.21-2-3

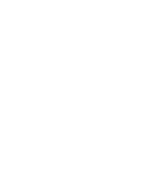


5-0.21-2-4



5-0.29-0.5-1

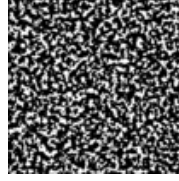
5-0.29-0.5-2



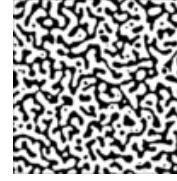
5-0.29-0.5-3



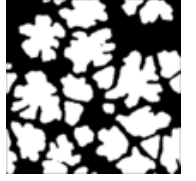
5-0.29-0.5-4



5-0.29-1-1

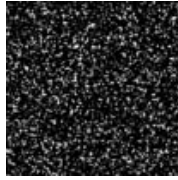


5-0.29-1-2

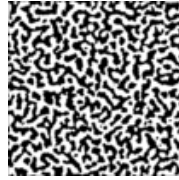


5-0.29-1-3

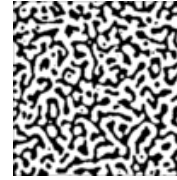
5-0.29-1-4



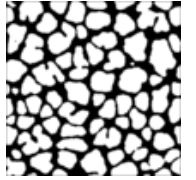
5-0.29-1.5-1



5-0.29-1.5-2



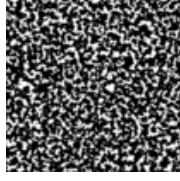
5-0.29-1.5-3



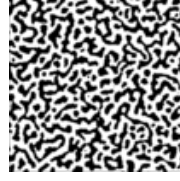
5-0.29-1.5-4



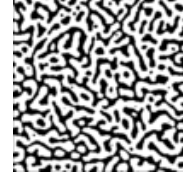
5-0.29-2-1



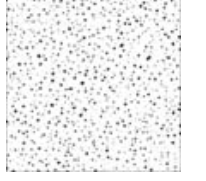
5-0.29-2-2



5-0.29-2-3



5-0.29-2-4



20-0.05-0.5-1



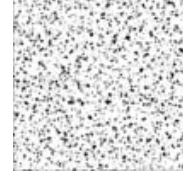
20-0.05-0.5-2



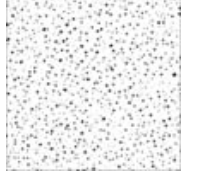
20-0.05-0.5-3



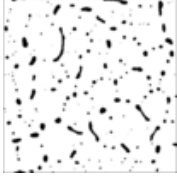
20-0.05-0.5-4



20-0.05-1-1



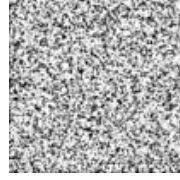
20-0.05-1-2



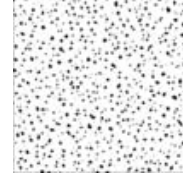
20-0.05-1-3



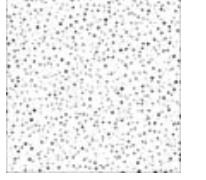
20-0.05-1-4



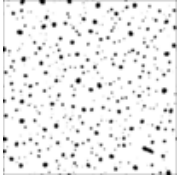
20-0.05-1.5-1



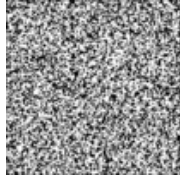
20-0.05-1.5-2



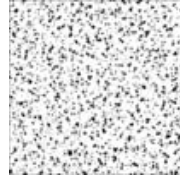
20-0.05-1.5-3



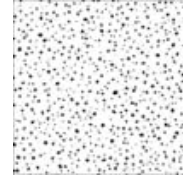
20-0.05-1.5-4



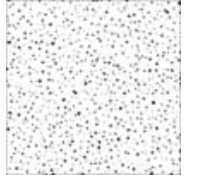
20-0.05-2-1



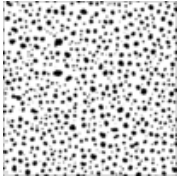
20-0.05-2-2



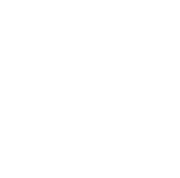
20-0.05-2-3



20-0.05-2-4



20-0.13-0.5-1



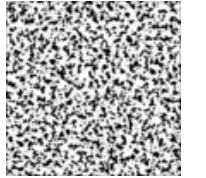
20-0.13-0.5-2



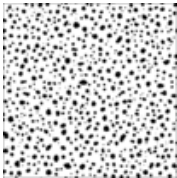
20-0.13-0.5-3



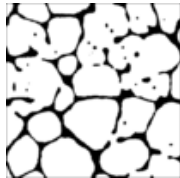
20-0.13-0.5-4



20-0.13-1-1



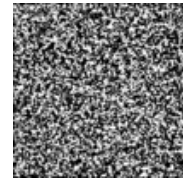
20-0.13-1-2



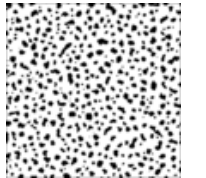
20-0.13-1-3



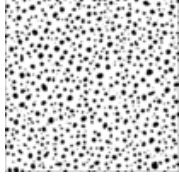
20-0.13-1-4



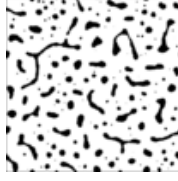
20-0.13-1.5-1



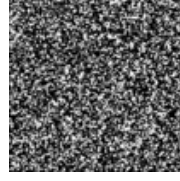
20-0.13-1.5-2



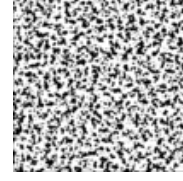
20-0.13-1.5-3



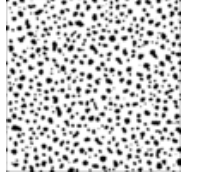
20-0.13-1.5-4



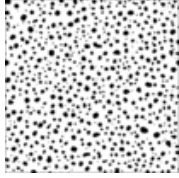
20-0.13-2-1



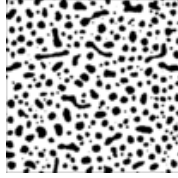
20-0.13-2-2



20-0.13-2-3



20-0.13-2-4



20-0.21-0.5-1



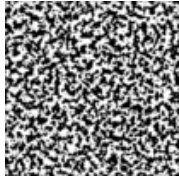
20-0.21-0.5-2



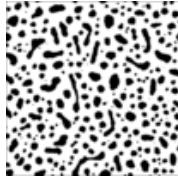
20-0.21-0.5-3



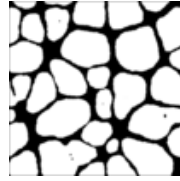
20-0.21-0.5-4



20-0.21-1-1



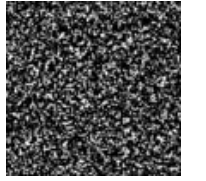
20-0.21-1-2



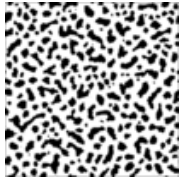
20-0.21-1-3



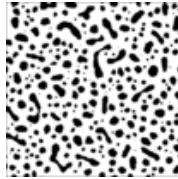
20-0.21-1-4



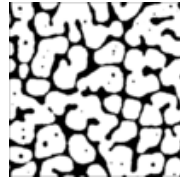
20-0.21-1.5-1



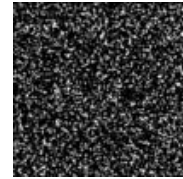
20-0.21-1.5-2



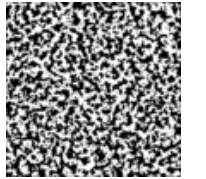
20-0.21-1.5-3



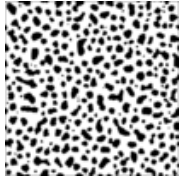
20-0.21-1.5-4



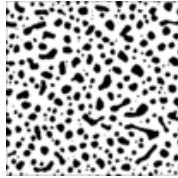
20-0.21-2-1



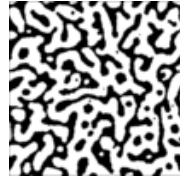
20-0.21-2-2



20-0.21-2-3



20-0.21-2-4



20-0.29-0.5-1



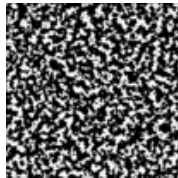
20-0.29-0.5-2



20-0.29-0.5-3



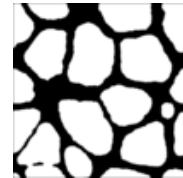
20-0.29-0.5-4



20-0.29-1-1



20-0.29-1-2



20-0.29-1-3



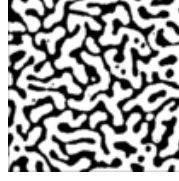
20-0.29-1-4



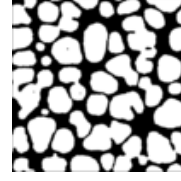
20-0.29-1.5-1



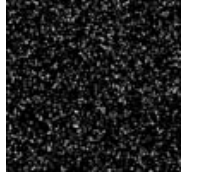
20-0.29-1.5-2



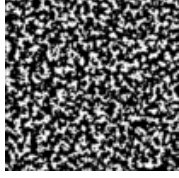
20-0.29-1.5-3



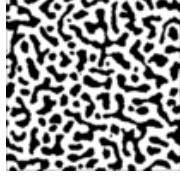
20-0.29-1.5-4



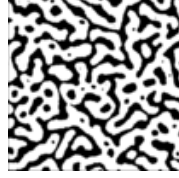
20-0.29-2-1



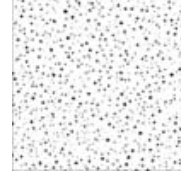
20-0.29-2-2



20-0.29-2-3



20-0.29-2-4



35-0.05-0.5-1



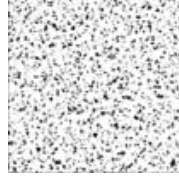
35-0.05-0.5-2



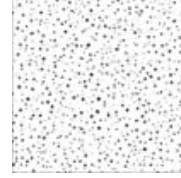
35-0.05-0.5-3



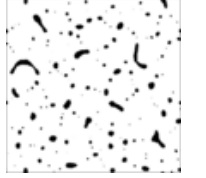
35-0.05-0.5-4



35-0.05-1-1



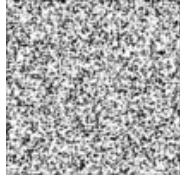
35-0.05-1-2



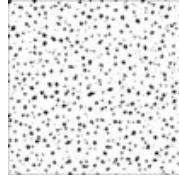
35-0.05-1-3



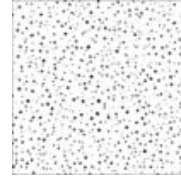
35-0.05-1-4



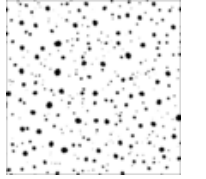
35-0.05-1.5-1



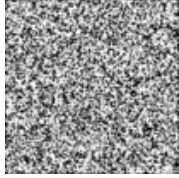
35-0.05-1.5-2



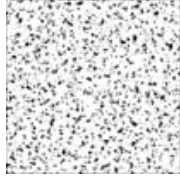
35-0.05-1.5-3



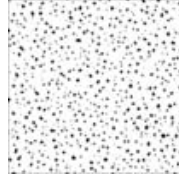
35-0.05-1.5-4



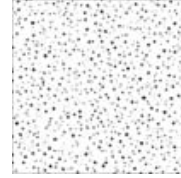
35-0.05-2-1



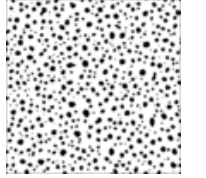
35-0.05-2-2



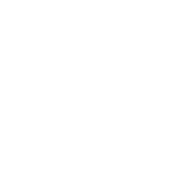
35-0.05-2-3



35-0.05-2-4



35-0.13-0.5-1



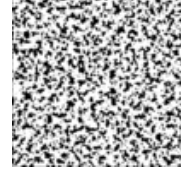
35-0.13-0.5-2



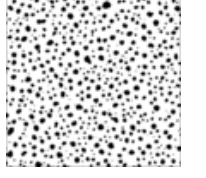
35-0.13-0.5-3



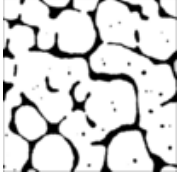
35-0.13-0.5-4



35-0.13-1-1



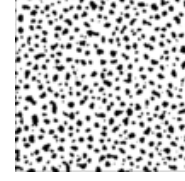
35-0.13-1-2



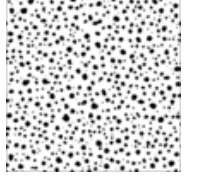
35-0.13-1-3



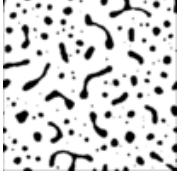
35-0.13-1.5-1



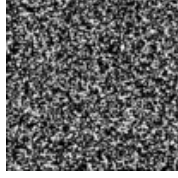
35-0.13-1.5-2



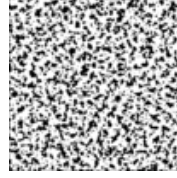
35-0.13-1.5-3



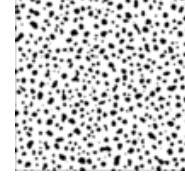
35-0.13-1.5-4



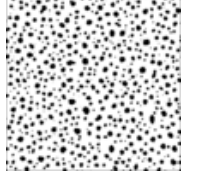
35-0.13-2-1



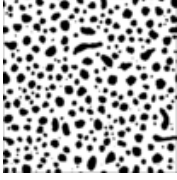
35-0.13-2-2



35-0.13-2-3



35-0.13-2-4



35-0.21-0.5-1



35-0.21-0.5-2



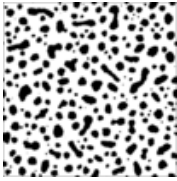
35-0.21-0.5-3



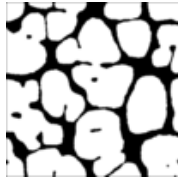
35-0.21-0.5-4



35-0.21-1-1



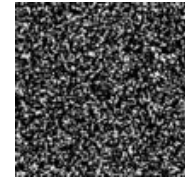
35-0.21-1-2



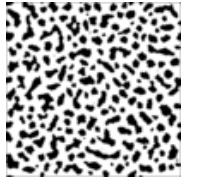
35-0.21-1-3



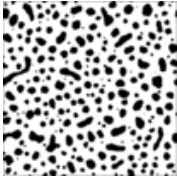
35-0.21-1-4



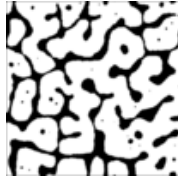
35-0.21-1.5-1



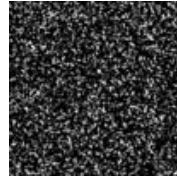
35-0.21-1.5-2



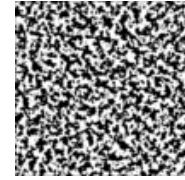
35-0.21-1.5-3



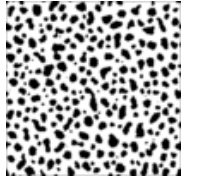
35-0.21-1.5-4



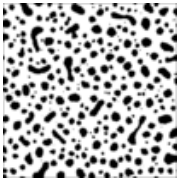
35-0.21-2-1



35-0.21-2-2



35-0.21-2-3



35-0.21-2-4



35-0.29-0.5-1



35-0.29-0.5-2

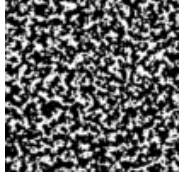


35-0.29-0.5-3



35-0.29-0.5-4

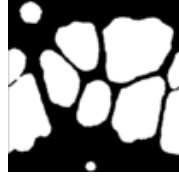




35-0.29-1-1



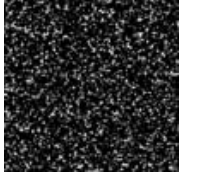
35-0.29-1-2



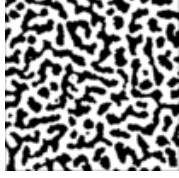
35-0.29-1-3



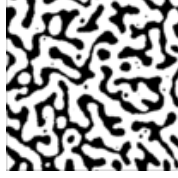
35-0.29-1-4



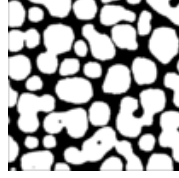
35-0.29-1.5-1



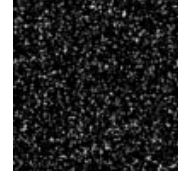
35-0.29-1.5-2



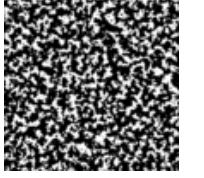
35-0.29-1.5-3



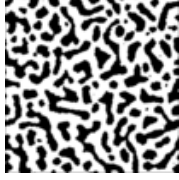
35-0.29-1.5-4



35-0.29-2-1



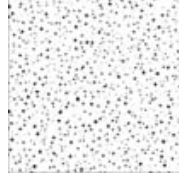
35-0.29-2-2



35-0.29-2-3



35-0.29-2-4



50-0.05-0.5-1



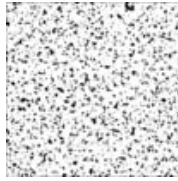
50-0.05-0.5-2



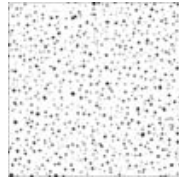
50-0.05-0.5-3



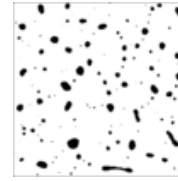
50-0.05-0.5-4



50-0.05-1-1



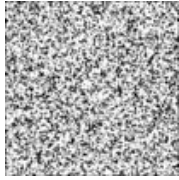
50-0.05-1-2



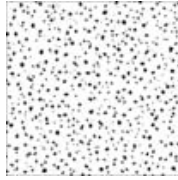
50-0.05-1-3



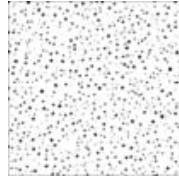
50-0.05-1-4



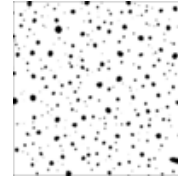
50-0.05-1.5-1



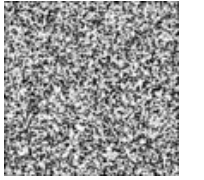
50-0.05-1.5-2



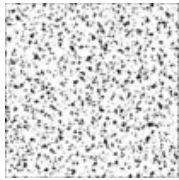
50-0.05-1.5-3



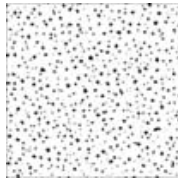
50-0.05-1.5-4



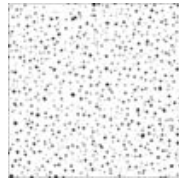
50-0.05-2-1



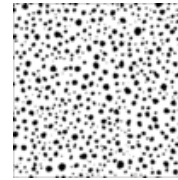
50-0.05-2-2



50-0.05-2-3



50-0.05-2-4

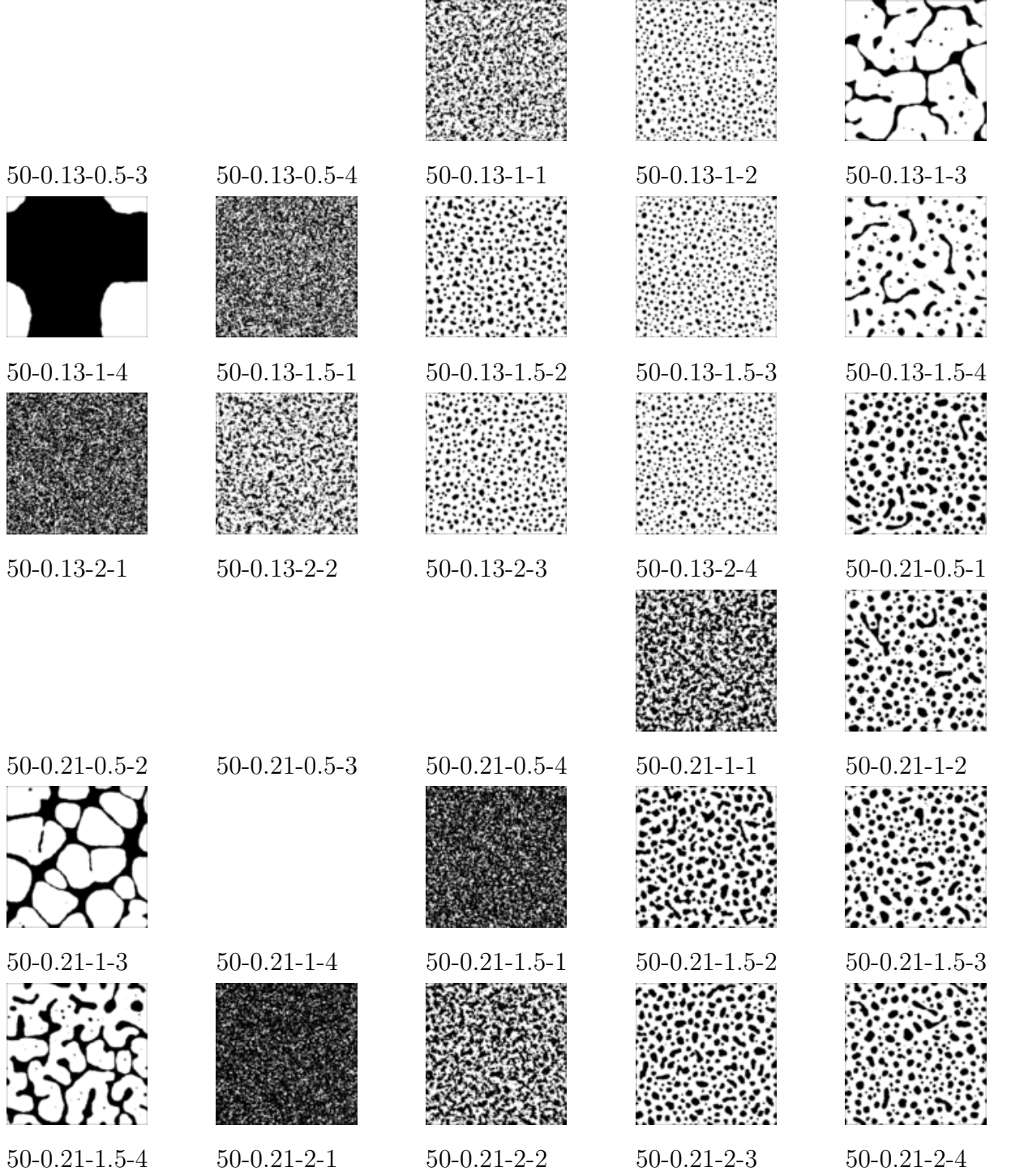


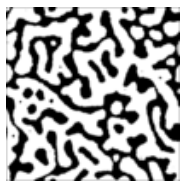
50-0.13-0.5-1



50-0.13-0.5-2







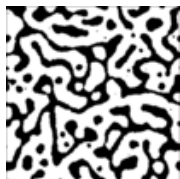
50-0.29-0.5-1



50-0.29-1-2

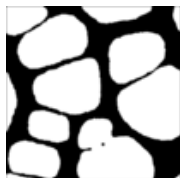


50-0.29-1.5-3

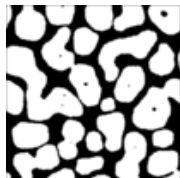


50-0.29-2-4

50-0.29-0.5-2

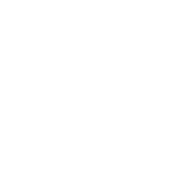


50-0.29-1-3

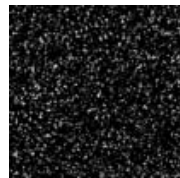


50-0.29-1.5-4

50-0.29-0.5-3

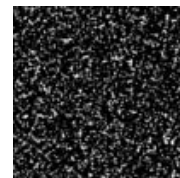


50-0.29-1-4

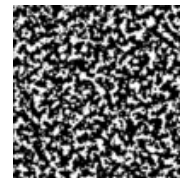


50-0.29-2-1

50-0.29-0.5-4



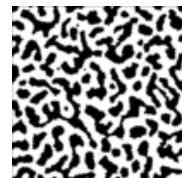
50-0.29-1.5-1



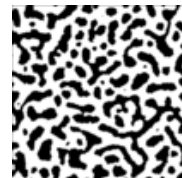
50-0.29-2-2



50-0.29-1-1



50-0.29-1.5-2



50-0.29-2-3